NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DEVELOPING A HIGH ASSURANCE MULTILEVEL MAIL SERVER

by

Bradley R. Eads

March 1999

Thesis Advisor: Cynthia E. Irvine

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 1999	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE : DEVELOPING A HIGH ASSURANCE MULTILEVEL MAIL SERVER	**5. FUNDING NUMBERS**
6. AUTHOR(S) Eads, Bradley R.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

Electronic mail (email) often contains sensitive information, which requires special handling to prevent accidental disclosure to unauthorized personnel. Using multiple systems operating at different classifications has caused a number of inefficiencies in the way email is managed and distributed. Attachments in emails have increased the likelihood that a "Trojan Horse" could be inserted in the system to obtain unauthorized access to information.

To address this problem Commercial Off-The-Shelf (COTS) software for Internet Message Access Protocol (IMAP), a mail server protocol, has been adapted to a high assurance multilevel base, the Wang Federal XTS-300. The XTS-300 constrains information flow permitting information at different sensitivity levels to be stored securely. Controlled access to mail is provided to client workstations. Enhanced with a trusted computing base extension, these COTS IBM PC compatibles run a standard office productivity suite. This architecture eliminates the risk that a "Trojan horse" will pass higher sensitivity information to a lower sensitivity level.

The research resulted in an IMAP server application running on the XTS-300 that managed the multilevel mailbox data structure using single level instances. Single level clients are then able to access mail through these instances.

14. SUBJECT TERMS Server, mail, networks, multilevel, communications	**15. NUMBER OF PAGES** 126
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK.

DEVELOPING A HIGH ASSURANCE MULTILEVEL MAIL SERVER

Bradley R. Eads
Major, United States Marine Corps
B. A., Wabash College, 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1999

Author: [illegible]
Bradley R. Eads

Approved by: [illegible]
Cynthia E. Irvine, Thesis Advisor

[illegible]
Daniel F. Warren, Associate Advisor

[illegible]
Daniel C. Boger, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK.

ABSTRACT

Electronic mail (email) often contains sensitive information, which requires special handling to prevent accidental disclosure to unauthorized personnel. Using multiple systems operating at different classifications has caused a number of inefficiencies in the way email is managed and distributed. Attachments in emails have increased the likelihood that a "Trojan Horse" could be inserted in the system to obtain unauthorized access to information.

To address this problem Commercial Off-The-Shelf (COTS) software for Internet Message Access Protocol (IMAP), a mail server protocol, has been adapted to a high assurance multilevel base, the Wang Federal XTS-300. The XTS-300 constrains information flow permitting information at different sensitivity levels to be stored securely. Controlled access to mail is provided to client workstations. Enhanced with a trusted computing base extension, these COTS IBM PC compatibles run a standard office productivity suite. This architecture eliminates the risk that a "Trojan horse" will pass higher sensitivity information to a lower sensitivity level.

The research resulted in an IMAP server application running on the XTS-300 that managed the multilevel mailbox data structure using single level instances. Single level clients are then able to access mail through these instances.

THIS PAGE INTENTIONALLY LEFT BLANK.

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK.

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK.

I. INTRODUCTION

A. PURPOSE

The Department of Defense (DoD) is confronted daily with the task of providing mission-critical and time sensitive information to the people who need it to perform their mission. Many times this information is classified and requires special handling to prevent accidental disclosure to unauthorized personnel. This special handling causes an even greater dilemma when it is applied to the area of automated information systems (AIS).

Sensitive material is labeled with a classification level (Confidential, Secret, Top Secret, etc.) that indicates who can access this information. The clearance of the individual who is accessing the data must be greater than or equal to the level of the data being accessed. Most of the systems currently in use are system-high; since there are not labels, they restrict access to all of the data at the highest classification level of any data on that system[1]. This means that a person with a clearance limited to a lower classification level cannot access the system, even if the desired information is intended to be at a lower classification, because all data on the system has effectively been labeled as if it has the highest classification.

This is required because the "typical DoD computer system" running the UNIX or Windows NT operating system does not apply labels, has no assurance of policy enforcement, and cannot differentiate between different levels of data. These systems limit access to certain files using access control lists which are intended to enforce a Discretionary Access Control (DAC) policy. In many systems enforcing a discretionary policy, the owner of the data decides who can see it. This method is inadequate when it comes to classified material. The nondisclosure policy used for classified material is mandatory. Once data is labeled as having a certain level of sensitivity, access to that data is based on an established access structure. DAC systems are not capable of supporting the DoD requirements for protecting classified material from disclosure to unauthorized personnel. This is due to an information flow problem to be discussed in detail later.

The absence of systems to enforce mandatory nondisclosure policies causes a number of problems:

- Equipment duplication--Multiple systems are required if more than one level of data is going to be managed.
- Excessive clearances--Personnel must be cleared to the highest level of the data on the system they are going to work on, despite the fact that the data they need may be at a lesser classification.
- Incoherent views--Data that is spread across multiple systems, due to different classifications, may give the user an incomplete picture of the situation and produce an erroneous action.
- Inconsistent content--Data contained on different systems that is intended to convey the same information may become inconsistent due to individual system updates and changes.
- Untimely information—The data may be delayed in reaching certain individuals because it must be placed into each system separately.

There are systems available that can eliminate these inefficiencies by enforcing a Multilevel Security (MLS) policy. MLS requires labels that are used to separate data into different levels on a single system. Users of different classification levels can access the system at their respective clearance level or below, thereby gaining access to the information they need. However, they cannot read or manipulate data that they are not cleared to access.

Now a brief example of how a nondisclosure policy can not be enforced in the "typical system." In a typical single level system, one user (Bob) can give another user (Alice) a program that performs some useful function. Alice can now use this program, but when it runs, it runs with her permissions. This means that it can perform any action on Alice's files that she could perform. So when she runs this program it also performs a background function; it changes the access protection on all of the files in Alice's directory and gives Bob full access to them. It could even send this information to Bob. This exploits the flexibility of DAC mechanisms to perform actions in the system that the user would not willingly permit. Software with a hidden malicious functionality is said to contain a "Trojan Horse".

A Trojan Horse could be embedded in a nice screen saver that is given to the user by a "friend." When it executes, this particular screen saver shows some really great pictures on the screen. Unfortunately, in the background, it is sending a copy of all the users data to an email account in a foreign country.

A MLS system is also susceptible to this attack at a single level, but it constrains the flow of information between levels. This provides confinement of data and gives us the enforcement that we are trying to achieve between levels (e.g., between secret and confidential) not necessarily between users. All users operating at a particular level will have already been assigned a clearance to that level, so any exploitation must be limited to that sensitivity level.

Email is routinely used to provide information throughout DoD and other organizations and is susceptible to the same problems as any other data; a user could receive an email with a Trojan Horse as an attachment and accidentally execute it. Email can be sent and received at different classification levels and must be protected from unauthorized disclosure. The purpose of this thesis is to investigate the viability of building a MLS Mail Server on a high assurance system, the XTS-300, produced by Wang Federal Incorporated.

B. RESEARCH QUESTIONS

Several questions are addressed by this thesis. They, more than the actual creation of an operating mail server, are the crux of this research.

1. Which mail server program would be best suited for users in a multilevel environment and can it be adapted to work on a system that has been evaluated at Class B3 under the Trusted Computer System Evaluation Criteria (TCSEC) [2]?

2. Will multiple running instantiations of the mail server program be required to maintain the multilevel properties of the system?

3. Is there additional unanticipated functionality that may be desired on a multilevel mail system?

4. How difficult is it to port existing mail server software to a system that has been evaluated to Class B3 system and can the software's functionality be maintained?

C. THESIS OVERVIEW

This thesis will begin with an examination of the current practices concerning sending and handling email in DoD. A few of the pitfalls of these practices are highlighted. Next, a few of the advantages of using a MLS email system are described. This is followed by a brief discussion of previous work in this area. Chapter III cites the requirements for a trusted mail server, provides a brief explanation of Simple Mail Transfer Protocol (SMTP) and then compares the two primary storage protocols, Post Office Protocol Version 3 (POP3) and IMAP4.

Chapter IV explains the specification for the trusted mail server and provides the details of the modification and installation.

D. EXPECTED BENEFITS OF THIS THESIS

The Multilevel Security Program cites the lack of MLS in DoD AIS as the cause of “significant operational vulnerabilities by limiting interoperability and data fusion” [3]. The information obtained from this thesis can be used by all government agencies that must move classified information through email. The architecture to which this thesis contributes will provide for the elimination of duplicate workstations used in processing classified information at different levels. It should improve the effectiveness of the users of those systems by allowing them to look at all of their email simultaneously, through a single login. This is more efficient than logging on to multiple systems and may allow the user to assimilate the data more effectively improving the decision making process. This system will also allow for the use of IT-21 compliant workstations and should require a minimal amount of training for use.

II. SECURING ELECTRONIC MAIL

A. OVERVIEW OF CURRENT PRACTICES

Currently, almost all classified systems used in DoD are operated at a single level, the highest level of any data being processed within the particular system. This is system-high mode and requires that all personnel who have access to the system be cleared to that highest level. Additionally, all outgoing email from that system must be handled at the highest level. This creates numerous inefficiencies: The entire system, not just the server must be maintained in a secure environment. Additional systems must be provided to allow users to send email at lower classifications. Separate networks must be managed for each sensitivity level.

Lower level data that must be read by users at a higher level must be moved to the higher level manually or using a one-way link. In the manual method, data is placed in the system by the physical transfer of media (disk, tape, etc.) from the lower system to the higher system. This is sometimes referred to as an "air gap." The manual method is a time consuming process that inhibits the near real time response of email.

Another method of providing access to lower level data at a higher level is to install a one-way data link that allows the low data to be transferred to the high level. This solves the response time problem and eliminates the administrative tedium of manually transferring data across the air gap. The Naval Research Laboratory Pump[4] is one such link. Another is the Information Diode that is available from Galaxy Computer

Services Incorporated. The Information Diode used an automated version of UNIX standard trivial file-transfer protocol (tftp) to move files from a low network to a high network [5]. These methods still create additional classified information that must be stored, monitored, declassified and disposed of, because without labels the system cannot differentiate the classified data from the unclassified data that was added.

B. ADVANTAGES OF A MULTILEVEL SECURE EMAIL SYSTEM

Multilevel Email Systems have numerous advantages over single level systems. They eliminate the administrative overhead required to maintain an "air-gap." This is accomplished by using labels to differentiate between sensitivity levels. All data are stored in objects, which are labeled by a high assurance security kernel (kernel). This labeling is based on the classification level that the user is operating when the information container, or more abstractly object, is created or by the classification level associated with the network source.

Once the objects are labeled, all accesses to them are mediated by the kernel. This is referred to as Mandatory Access Control (MAC) because the attributes of objects (i.e. their labels) are immutable and protected by the kernel. MAC is not enforced at the discretion of the owner of a file , as is DAC. The kernel enforces the DoD secrecy policy as described in the Bell and LaPadula Model [6]. Control of flow is now managed by the mandatory policy enforcement of the MLS.

Users with different clearances can use the same system at different session levels. For any particular object, only users with clearance to the system at the sensitivity level of the object or higher are able to gain access to it. The contents of the object are protected. This prevents disclosure of the data to unauthorized personnel. As long as the correct administrative policies for assigning users to the system are in place, unauthorized users should not be able to gain access to sensitive information.

The kernel referred to in the previous paragraphs is on a high assurance system. This is a system that has been evaluated to Class B3 or higher by the TCSEC. High assurance systems must meet the objectives of the Reference Monitor concept. A subsystem satisfies the Reference Monitor concept if it mediates all accesses of subjects to objects, is tamperproof, and is small enough to be subjected to tests and analysis to assure completeness [7]. In high assurance systems, the policy that the kernel implements has been formally modeled. This ensures that the policy being enforced is consistent and can be expressed in technical terms. So that a mapping of the policy to the specification can be used to build the system to ensure that system software is correct. The Trusted Computing Base (TCB) is structured to exclude code not essential to security policy enforcement[8]. This gives the user confidence that the kernel does not contain undocumented and unwanted functions (i.e. Trojan Horses). The TCB also provides guarantees that when an application, such as the mail server is placed on the system; it will be constrained by the policy enforcement mechanism.

C. NPS TRUSTED MAIL SERVICES

During research performed by Downey and Robb [9], they conducted analysis and design work to establish the viability of a High Assurance, Multilevel Secure Mail Server (HAMMS). Their groundwork has provided a System Specification that documents the design requirements for creating such a system. The primary emphasis of this work was on using existing commercial-off-the-shelf (COTS) components to reduce cost, development time, maintenance and training.

The Trusted Mail service is designed to meet the following functional requirements:

- It should support a COTS PC client that is running a COTS operating system, a TCB extension may be installed but should be transparent to the Mail Server.
- It must support unmodified COTS mail client software.

It should also meet the following non-functional requirements:

- Provide Multilevel Security for separation of sensitivity levels
- Allow Discretionary Access Control to separate the data of individual users at a sensitivity level
- Support the ability to audit the system
- Keep cost to a minimum
- Be reliable
- Not pay an undue penalty in the performance observed by the user

The final prototype system relies on additional research that is being performed at the Naval Postgraduate School on the MLS Local Area Network (LAN). The MLS LAN will allow the use of COTS software and hardware at the individual user workstations, while providing multilevel functionality to the user, leveraging the capabilities of the server. This system will use the XTS-300 as the server platform; in this case, the first server will be an email server, the subject of this paper. A Trusted Path which guarantees the identity of the workstation or user to the server and the identity of the server to the user [10], a "diskless-like" workstation that enforces object reuse [11], and a modified Network Interface Card that provides a trusted path extension are being developed as separate research projects.

A "diskless-like" workstation is required to support object reuse. In order to prevent the system from inadvertently transferring information between levels, the workstation must not be able to store data between sessions. This means that the disk drive on the workstation cannot be allowed to store data between sessions. The exact implementation will be the subject of future research, but some combination of read-only disk drives and a large virtual disk created from volatile memory (e.g. Random Access Memory (RAM) chips) is expected.

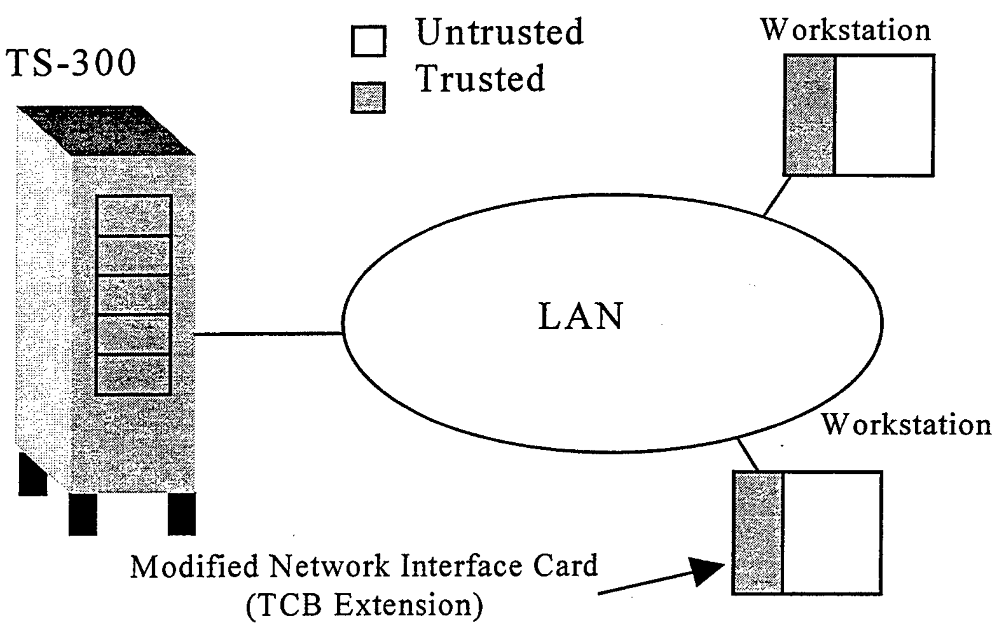

Figure 1 MLS LAN

The modified Network Interface Card, that implements an extension to the TCB, will have provisions to invoke the trusted path, enforce object reuse between sessions and communicate over the LAN. This may require the card to have a direct interface to the user, so that the trusted path can be invoked without intervention by the operating system. It will reboot the system or provide some other assurance between sessions to implement object reuse that is not inherent in the underlying operating system. The normal functions of a standard Ethernet network card will also be supported. Figure 1 gives a pictorial representation of the proposed system. Figure 2 represents the software contained on the XTS-300.

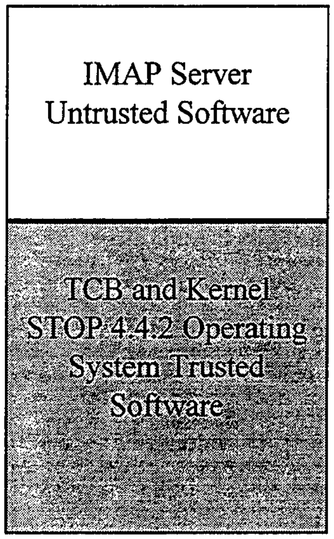

Figure 2 Software Residing on the XTS-300

THIS PAGE INTENTIONALLY LEFT BLANK.

III. A COMPARISON OF CANDIDATE MAIL SERVER PROGRAMS

A. OVERALL TRUSTED MAIL SERVER REQUIREMENTS

The Mail Server portion of this project has several requirements:

1. A user operating at a particular sensitivity level should be able to read all mail dominated by that sensitivity level (subject to DAC constraints).
2. A user operating at a particular sensitivity level should only be able to append to and send mail at that sensitivity level.
3. All mail read at a level should be marked as read at that level.
4. Mail attachments should be supported at the client using the COTS client mail software.
5. The server application must be constrained to the untrusted environment of the system on which it is operating.
6. It must provide for three levels of sensitivity and adhere to the TCSEC computer security requirements for open security environments.
7. It must store incoming email messages at different sensitivity levels as is appropriate to their label.
8. It must allow the user to add information to messages on the server and store draft messages on the server to simplify their management.
9. It should be as unobtrusive to the user as possible.
10. The ability to mark all messages that are read, even if they are lower at a lower access class will be implemented if time permits.

Each of these requirements is discussed in greater detail below.

Requirements 1, 2 & 3:

The first three requirements are based on the security requirements for mandatory non-disclosure. The users at a particular sensitivity level should be able to read mail at the same sensitivity level and lower. This allows users to see all mail that they are allowed to view based on sensitivity level and clearance, but prevents them from seeing any mail that they are not entitled to see (e.g., a user at confidential should be able to read confidential and unclassified email). Users at one sensitivity level should not be able to write to other sensitivity levels (e.g., a user at Secret should only be able to append information to Secret emails or send Secret emails). If the users were allowed to write down to a lower sensitivity level this would also allow hidden malicious software to exfiltrate data to lower levels. Writing up would not allow a unpermitted behavior, but would not make much sense since the user writing the data could not see where to place it. Mail messages should be marked as read at the users current sensitivity level since this is an expected behavior from a mail server and does not violate any policies.

Requirement 4:

Mail attachments are used on a regular basis by many email users. The attachments are made at the client, but must use a protocol that is supported by the mail server. The client software attachment protocol is the primary concern in achieving this requirement, because the attachment is added to the email message at the client and must be supported by the receiving mail server.

Requirement 5:

The mail server is going to be based on commercial off-the-shelf software that has been written by an outside source. We have not attempted to determine the correctness of the server software. Thus in order to maintain the integrity of our system this software must execute outside of the TCB. This is Ring 3 of the XTS-300's protection scheme where untrusted software resides. The software will not contain any trusted instructions and will be constrained by the operating system, STOP 4.4.2, that is installed on the XTS-300.

Requirement 6:

The requirement for at least three sensitivity levels allows the system to meet the *Guidance for Applying the DoD TCSEC in Specific Environments*, the "yellow book". The "yellow book" does not allow the system to provide protection for more than two different levels above the clearance level of the lowest users in a open security environment [12]. Open security environments require the most stringent security enforcement by the system. If this requirement is met, the system will provide protection against malicious users and the introduction of malicious software. Some examples of the three levels are: if the clearance level of the lowest users is unclassified, it can only contain data at sensitivity levels of unclassified, confidential and secret. If the clearance level of the lowest user is confidential, then the system can contain data at sensitivity levels of confidential, secret and top secret. Similar concepts could be applied to corporate sensitivity needs.

Requirement 7:

All email messages must be kept at the appropriate classification level. This means that they must be stored on the mail server, a high assurance system. The workstations will be operating at a single level and cannot be trusted to maintain the integrity of the labeling performed by the server or enforce policy.

Requirement 8:

The user must be able to store draft messages on the server as well as append and manipulate messages that have been received to simplify message management. This means that the server must do more than just store incoming messages. It must have provisions for creating additional directories and moving messages between directories. It must also have a mechanism for the user to store draft messages until they can be sent, all data on the workstation will be lost between sessions.

Requirement 9:

The Mail Server should also be easy to use and as unobtrusive to the user at the client as possible while enforcing the appropriate polices. This means that the functionality to the user should be very similar to that of an unclassified system. The user should get all email, at or below the current classification level with a single sign-on. At the current level, all actions performed by the user on the mail will be taken.

Requirement 10:

As an option, all email that has been read or deleted should indeed have that action taken. This option is a convenience for the user. It prevents the user from rereading previously read messages and facilitates the management of a users messages. These functions will require the system to set flags on files at a lower sensitivity level. Marking data as read or for deletion that is at a lower sensitivity level than the current session level requires a write-down, which violates our previous requirement for the *property, no-write-down.

The implementation of this feature would require all flags for messages in lower directories to be written to a file at the current level. This file would be a list of the messages and the appropriate flags. The user would then invoke the trusted path, inspect the list, and downgrade the information to the lower level.

In the case of the XTS-300, the sensitivity level is called the security level. The lowest security level available on the system is represented by 0 and the highest by 15 [13]. The level the user is operating at is negotiated with the system and any operations that the user performs can only read objects at or below that level and write objects at that level. The system administrator can create aliases that equate the numbered levels to standard human readable sensitivity levels such as “Proprietary Company Research Only”, “Personnel Data” or “Public”. For DoD/U.S. Government use one can equate secrecy level 0 to Unclassified, secrecy level 1 to Confidential and secrecy level 2 to Secret. The development of this system was not performed with “live” classified data. Different levels of data were simulated to test the results of the modification.

The XTS-300 also supports eight integrity levels, integrity level 0 through integrity level 7 [11]. The integrity level is used to classify the criticality or reliability of the software. System software such as the STOP OS is highly critical and reliable and runs at integrity levels above three while standard applications are installed at level three and below to prevent less reliable and critical software from writing into (contaminating) more reliable software.

B. COMPARISON OF CANDIDATE SERVER PROGRAMS

1. Mail Server

Before describing a trusted email server, a description of an email server is needed. First, a brief general description of how email is routed would be beneficial. When a user sends email it is placed into a standard structure described by RFC 822, Standard for the Format of ARPA Internet Text Messages [14]. This detailed document is 47 pages long and defines all aspects of an email message. In particular it details the header, which has required fields and optional fields. The required fields are “Date”, “From”, and “To”, “cc” or “bcc”. These are the times that the message was created, the sender and the recipients respectively.

The most common optional fields are “Received”, “Subject”, “Reply-To” and “Return-Path”. The “Received” field provides information about the routing of the message to the recipient and allows it to be traced. The “Subject” is a brief description of the subject of the message. This utility of this field can be seen by anyone using email; it allows the recipient to find out about the contents of the message without opening the whole message. The “Reply-To” field is the email address that the sender would prefer that you send any email. Finally, the “Return-Path” is the return address of the message.

There is no requirement to send these fields in a particular order. It is suggested that they be sent in the order “Return-Path”, “Received”, “Date”, “From”, “Subject”, "Sender", "To", "cc", "bcc", followed by additional optional fields. The specification permits multiple occurrences of many of the fields.

An example of a minimum header is:

```
Date:  25 Mar 99 1030 PST
From:  eads@cs.nps.navy.mil
To:    flynlow@aol.com
```

The email program that is used to create email messages generates all of these fields, with the exception of "To", "cc" or "bcc", without direct interaction by the user. The "To", "cc" or "bcc" address usually follows the Internet format of somebody@someplace.domain. Somebody is the recipient's user name and someplace.domain is the domain name of the mail server that is getting the email from the network. The domain name is automatically translated to an Internet Protocol (IP) address (i.e., 131.120.10.88) by a domain name server.

After the sender creates an email message, it must be sent to the recipient. This is most commonly performed using the Simple Mail Transfer Protocol (SMTP) as described in RFC 821[15]. This is a three-step process. Briefly, the sending computer sends a MAIL command. This tells the SMTP receiver that it should prepare to receive a new mail message. This is also how the reverse-path is built. The "From" field should eventually contain a list of all of the hosts that transferred the message as well as the originator of the message. If it is accepted, the receiver returns a reply of "250 OK".

The sender then transmits a RCPT command that gives one forward-path address. Multiple RCPT commands may be sent to indicate multiple locations, such as the intermediate host as well as the final recipient. If this command is accepted, the receiver returns a "250 OK" to the sender. If the destination contained in the RCPT TO is unknown to the receiver, it returns a "550 Failure". The third step is the DATA command, which indicates the start of the message text. The receiver sends a "354 Intermediate" reply to indicate that it will consider all following lines to be message text. The final line of the message is a line containing only a period, the end of mail data indicator. This denotes to the receiver that the mail transaction has completed and that it can now process the message. The receiver returns a "250 OK" reply if it accepted the message.

Below is a brief illustration of how this works. This message has been sent from me@myplace.domain to somebody@someplace.domain, them@someplaceelse.domain and noone@noplace.domain. First the communication path between the two servers is opened and then the message is passed. "##" indicates a comment for the reader.

```
Receiver    someplace.domain Simple Mail Transfer Protocol Ready
## The receiver can receive messages that comply with SMTP.

Sender      HELO myplace.domain
## Opening SMTP communication from myplace.domain.

Receiver    250 someplace.domain
## Someplace.domain is ready to receive.

Sender      MAIL FROM: <me@myplace.domain>
## Mail is coming from me@myplace.domain.
```

Receiver	250 OK
Accepted.

Sender	RCPT TO: <somebody@someplace.domain>
This message is going to somebody@someplace.domain.

Receiver	250 OK
Accepted.

Sender	RCPT TO: <@someplace.com:them@someplaceelse.domain>
You need to send this message on to them@someplaceelse.domain.

Receiver	250 OK
Accepted.

Sender	RCPT TO: <@someplace.com:noone@noplace.domain>
You need to send this message on to noone@noplace.domain.

Receiver	550 Location unknown
I cannot contact noplace.domain.

Sender	DATA
I am now sending the message.

Receiver	354 Start mail input; end with <CRLF>.<CRLF>
Ready to receive the message.

Sender	Mail Message to include header information is sent here.
Sending header information first.

Sender	Continue to send message text.
Sending the rest of the message.

Sender	. (There is only a period on this line.)
The entire message has been sent.

Receiver	250 OK
I have received the entire message.

This example shows that "someplace" has received the data and is placing it in a mailbox for "somebody". It is also preparing to forward it on to "them".

There are three services that we usually associate with email they are a User Agent, a Message Transfer Agent and a Message Store. [16] We usually think of these in terms of the common applications associated with them. The User Agent is the application used to create and read mail messages. On UNIX systems, this is commonly Mail or mailx, while on personal computers (PC) it is usually integrated into the browser (Netscape or Internet Explorer). Eudora is also a common user agent.

The Message Transfer Agent is the service that actually sends the mail message from one system to another. The common UNIX application is sendmail. In PCs, this function is also performed by the browser. The Message Store is a system for storing messages when the User Agent is not connected to the Message Transfer Agent. The most common Message Store is currently Post Office Protocol, Version 3 (POP3).

The Mail Server can then be defined as a Message Transfer Agent and a Message Store. Outgoing email messages are sent on to their destination after being processed by the server. Incoming email messages are held in mailboxes until they can be accessed by the user. This is precisely where a decision must be made concerning the High Assurance Multilevel Mail Server.

What storage protocol is most appropriate, the current standard, POP3, or Internet Mail Access Protocol, Version 4 (IMAP4)? Keep in mind that this is a discussion of protocols and not actual implementations of them.

2. POP3

Described by RFC 1939, Post Office Protocol – Version 3[17], POP3 has been accepted by the Internet Architecture Board Standards Track as a standard protocol and is in widespread use. In the words of the author of RFC 1939, it "is intended to permit a workstation to dynamically access a maildrop on a server host in a useful fashion." It is a download only protocol. This means that the user has two options: download and delete or just download. There are no provisions for performing any other actions to the mail while it is on the server.

There are three states and eight commands required to implement the POP3 protocol. Additionally, there are five optional commands that implement additional functionality and security. The commands are only applicable in certain states and will be discussed in that context.

After a Transport Control Protocol (TCP) connection has been made between a client and the server on port 110, a connection is established and an initial greeting is received from the server. The remainder of the session consists of client/server interactions. Client server interactions are composed of a client command, server data and a server completion response. All interactions are transmitted as lines followed by a carriage return and line feed (CRLF). The client reads a line or a known number of octets followed by a Carriage Return and Line Feed (CRLF).

The session is now in the AUTHORIZATION State. Next, the client must identify itself to the server. Three optional commands are provided; they are USER, PASS and APOP. QUIT may also be used while in any state.

- USER and PASS—USER and PASS must be used in conjunction and are similar to initial identification and authorization at system sign-on. USER is followed by a mailbox name (user name). A reply is received from the server. If USER was successful then the PASS command is sent followed by the user's password. Successful completion of this step allows entry into the TRANSACTION State. Unsuccessful completion of either step requires both commands to be given again.
- APOP—can also be implemented in POP3. The APOP command must be followed by a mailbox name and a MD5 digest. This command avoids sending the password over the network in the clear. It involves using a shared secret that is used to create one-time passwords based on the system clock.
- QUIT—if given in this state, the connection to the server is lost.

When in the TRANSACTION State the client is associated with a specific mailbox and can make requests using the STAT, LIST, RETR, DELE, NOOP, and RSET commands. The QUIT command is also available.

- STAT--returns the number of messages and their total size in octets.
- LIST--returns the same information as STAT followed by a listing that assigns each message a number and size in octets.
- RETR--is followed by a message number. If the command is successful, the message corresponding to the message number is sent to the client.
- DELE--followed by a message number, marks that message for deletion.
- NOOP—does nothing, but does return an OK from the server.
- RSET--removes the deletion marking from any marked messages.

- QUIT--moves the session from the TRANSACTION State to the UPDATE State. In this state the server deletes all messages marked for deletion and closes the connection. If the QUIT command has not been given in the TRANSACTION and the connection is broken the messages marked for deletion shall not be removed.

This brief synopsis of POP3 shows the simplicity of the design. It allows the client to download and delete messages. It does not allow the client to perform any other operations on the messages contained in the server. It also does not allow the client to upload messages to the server. Additionally, the user can have multiple mailboxes, but they must be accessed with different user names and passwords.

3. IMAP4

Internet Message Access Protocol, Version 4rev1 is an alternative to POP3 and is described by RFC 2060 [18]. It is a newer and more complex message store protocol; it is currently a proposed standard protocol not a standard protocol like POP3. It has gained some acceptance and is being supported by the new versions of the popular browsers (Netscape Communicator and Microsoft Internet Explorer). This protocol has a large number of commands, especially when compared to POP3. It allows the user on the client system to manipulate the mail messages while they are on the server. The user can create delete and rename mailboxes as well as messages. It is worthwhile to briefly examine the protocol for comparison.

The first consideration is that IMAP4 just requires a reliable data stream, not necessarily TCP. If TCP is used then the server listens on port 143. The interaction transmission protocol between the client and server is the same as in POP3, using lines followed by a CRLF or a known number of octets followed by a CRLF. The commands are prefixed with an identifier generated by the client. This identifier is usually a short alphanumeric string (e.g. B243 or x002). No explanation is given for this requirement. There are three possible responses to a client command: OK (the command was successful), NO (the command was unsuccessful) and BAD (the command was unrecognized or contained a syntax error). The protocol requires that "the client must be prepared to accept a response from the server at all times."

The ability to manipulate messages on the server creates additional requirements to ensure that messages are handled properly. A Unique Identifier (UID) Message Attribute generated by the IMAP server is added to each message. This is a 32-bit value that is assigned to the messages of a mailbox in ascending, although not necessarily contiguous, order. Each mailbox has a 32-bit Unique Identifier Validity Value. These two values are combined to form a 64-bit value that is permanently guaranteed not to refer to any other message in the mailbox. A message sequence number, analogous to POP3's, that orders the messages from 1 to the total number of messages in the mailbox, is also specified.

A list of flags may also be associated with a message. Each flag begins with "\". Currently six have been defined.

\Seen—the message has been read.

\Answered—the message has been answered.

\Flagged—the message is "flagged" for special attention.

\Deleted—the message has been deleted and will be removed later by EXPUNGE.

\Draft—the message is still being composed.

\Recent—this is the first session in which this message has been present.

These flags may be permanent (e.g. \Answered) or session only (e.g., \Deleted).

Four more attributes are kept with each message. The Internal Date Message Attribute contains the date and time the message was received by the server. The Size Message Attribute contains the number of octets in the message. The Envelope Structure Message Attribute is a parsed representation of the RFC-822 envelope information in the message. The Body Structure Message Attribute is a parsed representation of the MIME-IMB body structure information in the message.

Similar to POP3, IMAP4 defines four states and provides state specific commands. Three commands, CAPABILITY, NOOP, and LOGOUT can be used in any state.

- CAPABILTY--returns a listing of the capabilities the server supports.
- NOOP--always succeeds, because it does nothing.
- LOGOUT--closes the connection between the server and the client.

The Non-Authenticated State is entered immediately following the connection. In this state either the LOGIN or AUTHENTICATE commands can be used, depending on the implementation to enter the Authenticated State. After the client is authenticated, the

Authenticated State is entered. The IMAP4 protocol also has provisions for pre-authentication of the client. This allows the session to begin in the Authenticated State.

The Authenticated State allows the use of an additional eleven commands. These are commands that manipulate mailboxes.

- SELECT—allows the selection of a specific mailbox. Moves the session into the Selected State.
- EXAMINE—the same as SELECT except the specified mailbox is read only.
- CREATE—creates a mailbox with a name given after the command.
- DELETE—deletes the specified mailbox.
- RENAME—renames the specified mailbox to a new name as given. If INBOX is renamed all the messages in INBOX are moved to a new mailbox with the given name. INBOX is required to be present at all times.
- SUBSCRIBE—adds a specified mailbox to the server's list of active mailboxes.
- UNSUBSCRIBE—removes the specified mailbox from the server's set of active mailboxes.
- LIST—returns a subset of all the mailboxes available to the client. Wildcards may be used.
- LSUB—returns a subset of all the mailboxes that the user has declared as active using the SUBSCRIBE command.
- STATUS—requests the status of the indicated mailbox without affecting the currently selected mailbox. It does not accept wild cards. It requires a mailbox name and one or more data status items (MESSAGES, RECENT, UIDNEXT, UIDVALIDITY and UNSEEN). It can return the number of messages and/or the unique identifier validity value and/or the nest UID value to be assigned to a message.
- APPEND—places the literal argument as a new message at the end of the specified destination mailbox. This allows messages to be added to a mailbox.

The Selected State is concerned with the manipulation of messages in a specific mailbox. The 14 previously defined commands are available as well as eight new ones.

- CHECK—performs implementation specific housekeeping associated with the mailbox. If the specific implementation does not perform this housekeeping then it is equivalent to NOOP.
- CLOSE—permanently removes all messages with the \Deleted flag set and returns to the authenticated state (no mailbox selected).
- EXPUNGE— permanently removes all messages with the \Deleted flag set, but remains in the selected state. It also returns a status as each message is removed, making it more informative and time consuming than CLOSE.
- SEARCH—searches the mailbox for messages that match the specified search criteria. Thirty-six search keys have been defined and can be found in RFC 2060.
- FETCH—the similar to RETR in POP3. It retrieves the requested data elements associated with the specified message(s). Fifteen elements have been defined.
- STORE—used to update or change the flags associated with the specified message(s).
- COPY—copies the specified message(s) to the end of the specified destination mailbox.
- UID—used to perform COPY, FETCH and SEARCH by unique identifier instead of sequence number.

It is easy to see that the additional functionality of IMAP4 brings a great deal of complexity with it; 22 commands are the minimum required set to meet the needs of the protocol. The final state is the Logout State. During this state, the connection is being terminated. This state can be requested by the client or by decision of the server.

4. POP3 vs. IMAP4

The previous descriptions of the two storage protocols demonstrate the complexity differences between POP3 and IMAP4, eight versus 22 commands, respectively. They also highlight the fact that POP3 and IMAP4 have different uses. POP3 is designed as a temporary storage point until the messages can be downloaded to the user's workstation or saved in the user's file space. It is not intended for the user to store email messages on the server. IMAP4 has facilities for creating additional files on the server and accessing multiple mailboxes. It has provisions for saving messages to the server that can be sent later. It is expected that the user will store email messages on the server.

In this particular implementation, the fact that draft messages can be uploaded to the server without being sent may be the biggest advantage that IMAP4 holds over POP3. Since classified data cannot be stored on the workstation, any unsent messages would be lost when the user logged off. The only way around this would be to send draft messages to yourself.

This also affects the user's access to email. In POP3, if the user's messages are not stored on a file server, then access to them after they are removed from the server is limited to a single workstation. This can be overcome by just leaving the messages on the mail server. However, is does not give the client the requisite functionality. In order to provide access at multiple sensitivity levels the messages must be kept on the XTS-300, which provides the necessary assurance to keep the information separated. IMAP, since it is designed to store and manipulate email messages on the mail server, allows the user to access email messages from any workstation connected to the server.

C. SELECTION

IMAP4 has been selected as the mail server of choice to implement in the High Assurance Multilevel Mail Server. It provides the ability to perform all actions on email messages at the mail server. This will enhance the ability of the system by allowing users to share email messages, save drafted email messages on the server, and create multiple files to organize and track email messages that have been received.

Additionally, the University of Washington has made its IMAP4 source code, in C, freely available at its website, the IMAP Information Center [19] where this effort is headed by Mark Crispin. This source code has been ported to a large number of systems. It should be possible to port this code, with some modification, to the XTS-300.

THIS PAGE INTENTIONALLY LEFT BLANK.

VI. APPLICATION OF MAIL SERVER PROGRAM TO THE XTS-300

A. SPECIFICATION

The High Assurance Multilevel Mail Server will use the Trusted Computing Base supported by the XTS-300 to constrain untrusted instances of the mail server program at each classification level. This will allow the placement of a large untrusted mail server program on the system without having to establish its correctness. This also allows the current evaluation of the system against the Trusted Computer Systems Evaluation Criteria to remain in effect.

The following are requirements for the server.

- It should implement the standard IMAP functions as specified in RFC 2060.
- Modifications that make the server work at multiple levels should not affect the successful execution of client mail software, such as Netscape.
- The server must provide storage locations for data at different sensitivity levels.

While this may seem like a very short and incomplete list, the list of IMAP functions and the functionality provided by the program is extensive.

The user will be pre-authenticated by the XTS-300 prior to gaining access to the IMAP server port, so the user will invoke the IMAP server in the Authenticated State. The LOGIN and AUTHENTICATE commands will not be required, at least not in this initial implementation. The CAPABILTY, NOOP and LOGOUT commands should not be affected by the file structure of the XTS-300, since they just require standard responses from the server.

The remaining commands that can be called in the Authenticated State are subject to the multilevel file structure of the XTS-300. For example, the user should then be able to SELECT a mailbox and have access to that mailbox as long as the sensitivity level of the mailbox is less than or equal to the current access level. If the user creates a new mailbox using the CREATE command it will be created at the current access level. The user should not be able to APPEND to a mailbox at a lower sensitivity level, since this would violate the * property (no-write-down). A user should be able to LIST all mailboxes that reside on the system at or below the level the current session level of the server instance.

B. MODIFICATION REQUIREMENTS

The IMAP4 software was obtained online from the University of Washington and was written by Mark Crispin. It has been ported to over 30 different operating systems including Windows NT and several versions of UNIX. It was designed for use on single-level systems and includes all of the source code needed to compile it. This section describes the port of this software to the XTS-300.

Initially, the downloaded software was compiled on a Sun Microsystems Ultra, just to ensure that the downloaded software would compile. This was the same system that was used to download the software from the Internet. After the software was downloaded, it just had to be unzipped and untarred. After that the imap-4.4 directory that the tar file created was entered and the IMAP Toolkit Makefile was invoked with the proper command line option for the Sun Ultra. This compiled IMAP with no errors. A floppy disk was then used to transfer the source code to the XTS-300.

Since the STOP 4.4.2 operating system was based on UNIX System V Version 3 [20], there were two primary porting options for compiling the software. The first, the "sv2" option was designed for UNIX System V Version 2 running on an AT&T PC-7300. This was an incomplete port, but it should not have contained any function calls that are not supported by System V version 3. The second was "sv4" a generic System V Version 4 port that was complete, but could be expected to require system calls that were unavailable on the XTS-300.

Initially the "sv2" option was tried. This "make" failed for a number of reasons from missing header files (e.g. syslog.h) to missing standard system function calls (e.g. rename). The System V version 4 default "make" file was then tried and it failed for many of the same reasons. At this point it was realized that a new option would have to be provided in the makefile for the XTS-300 and the STOP 4.4.2 operating system (OS).

The IMAP Toolkit Makefile uses multiple lower level make files to build the program. OS specific calls are made to a lower level C-client makefile; this makefile makes the differentiation between the various versions of the operating system. In this case since the operating system is close to UNIX the path to this makefile is "/src/osdep/unix." This path is invoked by the IMAP Toolkit Makefile based on the command line option chosen at compile time. All the specific dependencies are then handled by OS version specific header and source code files contained in the directory specified by this path. The os_sv2.c and os_sv2.h files were used as templates to create new .h and .c files. These were renamed os_xts.c and os_xts.h respectively.

The appropriate option for the XTS-300 then had to be added to the IMAP Toolkit Makefile. The name chosen for this option was "xts." The primary problem at this level was that the default compile used symbolic links. The STOP OS does not have the symbolic link option. The compiler would not run with symbolic links turned on, so the "-s" option, symbolic links, had to be removed from the link command. This required a change to the top-level make file. The changes made to this makefile were:

```
xts:
    $ (MAKE) ua LN=ln
    $ (BUILD) OS=$@ LN=ln
```

The entire IMAP Toolkit Makefile can be found in Appendix A.

The next step was to modify the C client makefile for the xts. The changes made to this file were:

```
xts: # XTS-300
    @echo Now compiling for XTS-300!!
    $ (BUILD) OS=$@ SIGTYPE = sv4 LOGINPW=std \
    MAILSPOOL=/usr/mail \
    RSHPATH=/bin \
    BASEFLAGS="-g –Dconst= -DSYSTEM5 –DSHORT_IDENT –
    I/usr/ethernet/include" \
    BASELDFLAGS="-lcass –lsocket –lc –lgen" \
RANLIB=true
LN=ln
```

The primary change here concerns the "-lcass" library option. CASS is the Commodity Application System Services library. It "provides an environment on the XTS-300 that is sufficient to allow the execution of UNIX-based commodity applications [14]." The additional path of –I/usr/ethernet/include was added and the RSHPATH was changed to "/bin", which is where rsh is located on the XTS-300. The rest of the option was taken from the sv2 option already contained in the makefile. A copy of the C client makefile is contained in Appendix A.

The os_xts.h file required modification to include .h files that were not required by other OS's but were required by STOP or that were located in a different directory. The included .h files were sufficient to compile the IMAP toolkit, but it may be possible to trim it to a necessary list if time permits. The following files were included: stdlib.h, sys/types.h, sys.dirent.h, errno.h, time.h and utime.h.

One .h file syslog.h was removed because application-initiated system level logging is not allowed in the untrusted environment on the XTS-300. This is discussed further when the logging events are discussed.

Two .h files that are needed to compile IMAP are not supported by STOP they are ustat.h and utime.h. These are standard files contained on UNIX System V workstations. A modified copy of each of these files was placed on the XTS-300 with no adverse affects to the system. A copy of each of these files is included in Appendix A.

Since the system will not allow applications running in the untrusted environment to make entries into the audit log, all logging events had to be terminated. The logging events are defined in os_xts.h, but they perform no function. System level auditing is a trusted process that is based on events that occur on the system. The OS on the XTS-300 provides 73 different events that may be selected by the system administrator for audit. The occurrence of these events will then generate an entry in the audit log. In particular, both login and logout are auditable events as well as the creation, deletion, opening and closing of any object. These and other auditable events should more than make up for the inability of an application to generate audit logs.

A number of functions were removed from os_xts.h because they were provided by the standard library or some other .h file. They were *getenv, memmove, strstr, strerror, strtoul, fsync, malloc, free* and *realloc*. After all these modifications to the os_xts.h file were made the file compiled without generating any errors.

Converting os_sv2.c into os_xts.c also took number of changes. The files flock.c and tz_sv4.c would not compile correctly on the XTS-300. The line which returned an integer to the calling program would not compile in flock.c. I had to explicitly cast the variable as an integer. The actual change was from "&fl" to "(int)&fl". The new file was then named flockxts.c, so that no changes to the original IMAP Toolkit were required.

The tz_sv4.c file was supposed to append the local timezone name to the input destination string. The compiler on the XTS-300 could not find the tzname variable that was required by the function. I rewrote the function with an appropriate workaround. The new file with the working function is called tz_xts.c. The renamed files are now called by os_xts.c instead of the originals; they can be found in Appendix A.

The functions *syslog*, *openlog* and *initgroups* were also required by the IMAP Toolkit, but unavailable on the XTS-300. All three of these functions were written in the os_xts.c file, so that they could be called by the program. They return true to let the program continue without failing, if they are called. The function *ftruncate* was needed by the IMAP Toolkit. I wrote a functional version in os_xts.c based on the standard UNIX version of this function. The files os_xts.h and os_xts.c are contained in their entirety in Appendix A.

Once these modifications were made to the IMAP Toolkit, I tried to compile and link the IMAP server once again. This time the program compiled with no errors. There were still numerous warnings, but most of these are due to concerns about how the compiler was interpreting "=" versus "==" and did not affect the final executable file.

It was now time to see if the executable that was produced by the IMAP Toolkit would work on the XTS-300. I then changed to the directory which contained imapd, the executable IMAP daemon that would eventually be installed on the server. I typed in "imapd" and the system started!! I was running on the XTS-300 at security level 0 and integrity level 3. These are the standard levels and are analogous to running at unclassified.

Upon execution the system returned the startup message:

```
* PREAUTH holmes.astro.cs.nps.navy.mil IMAP4rev1 v11.241 server ready
```

At this point I began to enter the commands that the server was expecting. The following three client commands comprise a representative sample of the commands given and the server responses. These are client-server level commands and would be hidden by the client program running on the workstation (e.g. Netscape):

```
1  Client:    a001 list "" *
                     ## List all mailboxes available to this client

   Server:    * LIST (\NoInferiors \UnMarked) "/" newmail
              * LIST (\NoInferiors) NIL INBOX
              a001 OK LIST completed
              ## The available mailboxes are "newmail" and "inbox".

2  Client:    a002 select inbox
              ## Select inbox to get mail off of the spool.

   Server:    * 6 EXISTS
              * NO Mailbox  vulnerable – directory must have 1777 protection
              * 6 RECENT
              * OK [UIDVALIDITY 917901723] UID validity status
              * OK [UIDNEXT 84] Predicted next UID
              * FLAGS  (\Answered \Flagged \Deleted \Draft \Seen)
              * OK [PEMANENTFLAGS (\*\Answered \Flagged \Deleted \Draft
               \Seen)] Permanent flags
              * OK [UNSEEN 1] 1 is first unseen message in /usr/mail/eads
              a002 OK [READ-WRITE] SELECT completed
              ## Inbox contains six messages all are new and none have been read.  It is
              open for read-write access.

3  Client:    a003 select newmail
              ## Select "newmail" from my workspace.

   Server:    * 4 EXISTS
              * NO Mailbox vulnerable - directory must have 1777 protection
              * 4 RECENT
              * OK [UIDVALIDITY 919374784] UID validity status
              * OK [UIDNEXT 13] Predicted next UID
* FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
* OK [PERMANENTFLAGS (\* \Answered \Flagged \Deleted \Draft
\Seen)] Permanent flags
* OK [UNSEEN 1] 1 is first unseen message in usr2/unclass/eads/newmail
a003 OK [READ-WRITE] SELECT completed
## Newmail contains four messages all are new and none have been read.
It is open for read-write access.
```

These three examples give the reader an idea of the type of information that is passed between the client and server programs without intervention from the user. All of the other IMAP commands were tested at this level and performed as expected.

After testing all of the IMAP commands at the 0 level, I renegotiated to level 1 and started testing all of the IMAP commands again. I quickly found a severe problem. The server could not CREATE or SELECT any mailboxes. This was due to the fact that all of the directories I was currently using were at level 0. The MAC implementation on the XTS-300 was working just as it should have. I could not CREATE a new mailbox or SELECT a mailbox for read-write because this would be a write down from level 1 to level 0.

This problem seems easy to fix: just create a directory for each level using the administrative commands provided on the XTS-300. It was not quite that easy. First, the IMAP server program was designed for single level systems and had no provisions for using different directories for different levels. Second, decisions had to be made as to how these directories would be organized. I had to figure out the second problem first. How should the directories be organized.

I had to have a different directory for each sensitivity level. Since I was planning to support three levels unclassified, confidential and secret, I needed three directories that corresponded. These were named “unclass”, “conf” and “secret” respectively. The question was then how should the path be set up? The natural inclination would be to set up the three directories for each user; the paths could be “usr2/username/unclass”, “usr2/username/conf” and “usr2/username/secret”. This implementation is very easy with respect to modifications of the server. It only requires the concatenation of the final directory onto the users pathname.

The problem is that all subdirectories on the XTS-300 that are at a higher sensitivity than their parent directory must be created by trusted processes and the IMAP server is not one. The system administrator would have to manually create all of the “secret,” “conf” and “unclass” directories that were required. This would be a minor inconvenience if only a few users were on the system, but a thousand users would require three thousand additional directories. A better solution is to insert the extra directory above the “username”. The directory paths then become “usr2/unclass/username”, “usr2/conf/username” and “usr2/secret/username”. The IMAP server running at the appropriate level can then add the username directory in each path without any attention from the system administrator.

The modification was made to the env_unix.c file. The following code was inserted to give the server knowledge of what level it is operating on, so that it can use the correct directory path.

```
/*****************************
    Added Code
*****************************/
 access_ma *user_level;          /*access_ma is a structure used to define mandatory
   access parameters. It contains a variable for security_level, integrity_level,
   integrity_catagories, and security_catagories.*/
 char *null_path = NIL;          /*Used to supply NIL to function getlevel()*/
 char slash[] = "/";             /*Used to supply "/" for directory path*/
 char *add_dir = '\0';           /*Used to store additional path name*/
 char secret[] = "secret";
 char conf[]  = "conf";
 char unclass[]= "unclass";
 char tmp2[MAILTMPLEN];
 getlevel(null_path, user_level);

 if(user_level->security_level==2)         /* Are we at the secret level?*/
 {
   add_dir = secret;                       /*Add the secret path*/
 }
 else if(user_level->security_level==1)  /* Are we at the confidential level?*/
 {
   add_dir = conf;                         /*Add the conf path*/
 }
 else                                      /* Everything else is unclassified*/
 {
   add_dir = unclass;                      /*Add the unclass path*/
 }
 int ix=0;                                 /* counter for loop*/
 while (*home && (home[ix] != '/'))       /*Loop until the slash*/
   {
          tmp2[ix] = home[ix];
          ix++;
       }       /*tmp should hold the first path in the directory*/

    strcat (tmp2, slash);  /* Add / to path*/
    strcat (tmp2, add_dir);  /* Add additional dir to path*/
    strcat (tmp2, slash);  /* Add / to path*/
    strcat (tmp2,user);/* Complete the path with the user's directory*/
  /*************************************************************/
```

This code looks at the level the server is currently operating at and places the appropriate directory in the pathname. After this modification was made the IMAP server was able to perform all functions at the different levels.

C. DIFFICULTIES

The compiling difficulties that were encountered during this thesis were mainly due to the peculiarities of the STOP OS. System V version 3 is not a widely used version of UNIX. Most systems either use version 2 or version 4, the choice of version 3 during system design hampered my progress and will hamper porting any commercial software to the XTS-300. I am sure that this was not a conscious decision on the part of the design team, but rather a result of the long lead-time required to develop high assurance multilevel systems. Version 3 was probably the latest version of UNIX when the system was being developed. The designers would have had no idea that it would also be one of the least used versions.

V. CONCLUSIONS AND FUTURE WORK

A. DISCUSSION

Porting application software to a trusted high assurance system is not as difficult as might be expected from a pure compiling and linking perspective. There was a large learning curve initially with respect to the STOP 4.4.2 OS, the IMAP Toolkit, the compiler for the XTS-300 and the structure of "make" files.

1. STOP 4.4.2 OS

The STOP 4.4.2 operating system is very similar to UNIX in some respects, but the differences are extreme. The system uses a Secure Attention Key (SAK) to negotiate a trusted path, which must then be used to perform numerous system level calls such as login, set level (sl), file system management (fsm) and logout (logout). This environment is very unforgiving of user errors and in the current age of Graphical User Interfaces (GUI's) seems to be somewhat of an anachronism. This requires the user to learn a completely new set of commands and interface techniques as compared to other systems.

2. The IMAP Toolkit

The IMAP Toolkit was well thought out and constructed in modules so that it could be easily ported to numerous systems. These same modules became a deterrent to finding methods to change the server to recognize multiple levels. A single function call could require tracing through four or five modules to determine how the data structure it contained was created and how it was being manipulated by the function.

It takes an extraordinary amount of time to become familiar enough with the IMAP Toolkit software to make the correct changes. The Toolkit consists of thousands of lines of code and the dependencies become difficult if not impossible to track. The original author, Mark Crispin, could probably offer additional insights into a more functional porting of the Toolkit.

3. The Compiler

The only compiler for the XTS-300 is a High C compiler from Metaware. Fortunately, the IMAP Toolkit was written in C. Unfortunately, many of the functions and libraries contained on the XTS-300 are documented so that correct usage is difficult. Some functions are completely undocumented. A perfect example of the inconsistency in documentation is the location of the information about additional libraries.

The XTS-300 has two libraries that must be called by the linker depending on whether the code is trusted or untrusted. Untrusted code must use the –lcass linker option, which calls the Commodity Application Services library, while trusted code must use the –oss linker option, which calls the Operating System Services (OSS) library. The –oss requirement is specifically mentioned on page one of the *Trusted Programmer's Manual* during the introduction. The requirement for -lcass is buried on page 67 of the

Application Programmer's Manual under the heading of intro(2) in the System Calls chapter.

While this is just one example this type of problem occurs time and time again. It adds to the work of writers of original software as well as people trying to modify existing software for use on the XTS-300. Improving the documentation that accompanies the XTS-300 would go a long way toward making it a more usable system.

4. Make files

All of the programming classes here at the Naval Postgraduate School are taught on PCs using a graphical integrated development environment. These systems build the "make" file as a background function and the user never sees it. Some of the higher level classes require programming on the UNIX system, but even in these classes the use of "make" files is kept to a minimum. This made the switch to the UNIX programming environment with a very intricate and complicated set of "make" files very difficult. Combined with the previous discussion of the compiler challenges, at times it was hard to determine if the problem was in the code or in the "make" file.

5. Goals Achieved by the Port

Porting the IMAP Toolkit to the XTS-300 to provide an IMAP4 version 1 server was the right decision with respect to our requirements for maintaining email on the server. It meets the requirements stated previously in this paper. The actual testing of a client email program is constrained by the lack of appropriate multilevel network services. All testing must be performed using the low-level email server commands as were demonstrated in Chapter IV of this document.

☐ A user operating at a particular sensitivity level should be able to read all mail dominated by that sensitivity level.

× A user operating at a particular sensitivity level can read all email at that particular sensitivity level and email in the directories of other users, provided the permissions are set correctly, that are dominated by that sensitivity level. The problem of reading email at all dominated sensitivity levels is a peculiarity of the software and is solvable given additional time. The user cannot read mail from multiple levels off the mail spool. The directory structure of this multilevel OS can get quite complicated. It contains a provision for "deflection directories" that automatically create a single level directory at the level of the data being inserted. By doing this a deflection directory is able to account for all levels on the system, but it can only be read by processes at a single level. Since the mail spool is a deflection directory it can only be read at a single level and does not support the read-down capability of the multilevel system.

☐ A user operating at a particular sensitivity level should only be able to append to and send mail at that sensitivity level.

√ A user operating at a particular sensitivity level can only append mail at that sensitivity level. The ability to send mail is constrained by the lack of completed network services, but mail sent to users on the XTS-300 is spooled at the correct level.

☐ All mail read at a level should be marked as read.

√ Mail that has been read using the low-level commands is marked as read.

☐ Mail attachments should be supported at the client using the COTS client mail software.

× This feature remains untested because the attachments must be performed at the client. The functionality to support MIME is supported by the IMAP4 v1 server, so this should work when the opportunity to test it arises.

☐ It must be constrained to ring three of the system on which it is operating.

√ The IMAP software was compiled and tested in ring three and was never placed in lower ring during the port. It is constrained by the STOP OS and cannot perform trusted functions.

☐ It must provide for three levels of sensitivity and adhere to the TCSEC computer security requirements for open security environments.

√ The software as currently ported will support three levels of sensitivity. The system is capable of supporting up to 16 secrecy levels if desired, but this would exceed the guidelines of the "yellow book."

☐ It must store incoming email messages at different sensitivity levels as is appropriate to their label.

√ This functionality is provided at the mail spool using a deflection directory. A message can be transferred to a new mailbox that must be at that same level.

☐ It must allow the user to append messages on the server and store draft messages on the server to simplify their management.

√ The APPEND function of the IMAP server works and can be used to append information to an existing message. Draft messages can be save in a mailbox until they are sent.

☐ It should be as unobtrusive to the user as possible.

× It was not possible to test this somewhat subjective requirement without a client.

☐ The ability to mark all messages that are read, even if they are lower will be implemented if time permits.

× Due to time constraints, this feature was left for future work.

6. MLS Service Solutions

Kang, Froscher and Eppinger developed requirements for MSL service solutions that could be expected to be of assistance in this project[21]. These requirements are:

- Independence from MLS architecture.
- Security unaware software.
- Efficient system resource utilization.
- Easy to manage.

These requirements sound enticing, but in practice are not feasible. The current MLS systems, of which there are few, all use proprietary operating systems. This requires all software used on those architectures to be modified in order to function.

Independence from architecture could only be achieved if a common operating system with some type of changeable hardware interface layer were developed.

The software that is placed on the system must be security aware. At least it must be aware of the multiple levels. The major modifications that were required during this work were modifications to make the software work at multiple levels. This was also the greatest hindrance. Software that is written for single level systems does not necessarily place the functions that must be modified to achieve multilevel operability in easily accessible locations.

System resource utilization is dependent on the operating system more than the application software. Our method of placing the MLS services on one system and the user interface layer on another system is the best solution with the current MLS systems. This also gives the designer more freedom at both levels. The server can be designed for speed and efficiency and the workstation can be designed for functionality and ease of use.

Management will always be more difficult in the MLS environment. Managing multiple levels can never be as easy as managing a single level. Additionally, high assurance multilevel systems are designed to have a minimalist kernel and TCB to make it easier to provide the necessary assurance. Without numerous additional layers of abstraction, this interface will continue to be difficult to use.

B. FUTURE WORK

Additional work on the High Assurance Multilevel Mail Server is required for it to be a viable system. The ability to write-down the flags for read and delete using a trusted application would make it possible to read all mail from the highest sensitivity level. It would require changing to lower levels only to send mail at those levels. One possibility for this would be to create a list of files to mark as read or deleted at the current level. After the user has logged off the trusted application could check for the existence of this file and compare the names of the files contained in it with the names of files at the lower level. If these files existed, the write-down could take place. Great care must be taken during the creation of this process to ensure that this overt channel does not create an exploitable covert channel.

Currently, the IMAP server is unable to retrieve mail at multiple levels from the mail spool. Combining the work of Downey and Robb [9] with the work in this thesis could be used to create a daemon that would copy mail from the spool into the user's IMAP directory, ensuring that all mail at all levels would immediately be available from the highest level. This would make the system more user friendly.

Application level auditing would be a great benefit for the management of the Multilevel Mail Server. The system level auditing of the XTS-300 provides extremely fine grained and low level auditing that will quickly fill the audit log and be tedious to review. The IMAP server software already contains calls to make entries to an audit log, if a mechanism could be built to capture the output of these calls to a log it would enhance the ability of the system administrator to glean useful information about the IMAP server.

C. CONCLUSION

When mandatory nondisclosure policies must be supported but multilevel secure systems are unavailable a number of problems result: equipment duplication, excessive clearances, incoherent views, inconsistent content and untimely information. These problems all stem from the need to provide separation data at different sensitivity level. Additionally, the threat of a Trojan Horse has only exacerbated the problem.

In this thesis a COTS mail server, IMAP, was ported to a high assurance MLS, the Wang XTS-300. The high assurance system was used to constrain the software and maintain separation between level. The IMAP server was used to create an interface to commercial client software.

The IMAP server runs on the XTS-300 and functions in the same way it would on a standard UNIX system. It was tested at all levels using the standard IMAP functions on the XTS-300. The software will perform all of the IMAP functions with respect to actions authorized by MAC (i.e. files can not be created at a lower level) with the exception of SELECT. Files selected at a lower level must be opened as read only. This has not been implemented. The system has not been used to host a client system, because the client server interface is incomplete.

APPENDIX A: GLOSSARY

Commercial-Off-The-Shelf (COTS)—refers to systems that are available from commercial venders and can be used without any modifications.

Discretionary Access Control (DAC)—a means of restricting access to objects based on the identity of subjects and/or groups to which they belong [2].

High assurance—systems that are evaluated to Class B3 or higher. Guarantees or provides confidence that the security policy has been implemented correctly and that the protection-relevant elements of the system do, indeed, accurately mediate and enforce the intent of that policy.

Internet Message Access Protocol (IMAP)—an email storage protocol that is designed to allow users to access and manipulate message on a server [20].

Mandatory Access Control (MAC)—a means of restricting access to objects based on the sensitivity of the information contained in the objects and the formal authorization of subjects to access information of such sensitivity [2].

Multilevel Security (MLS)—the ability of a system to contain information with different sensitivities that simultaneously permits access by users with different security clearances and needs-to-know, but prevents users from obtaining access to information for which they lack authorization.

Nondisclosure policy—a policy that allows only authorized individuals to have access to data a different levels of sensitivity.

Object—a passive entity that contains or receives information [2].

Open security environment—configuration control does not provide sufficient assurance that applications are protected against the introduction of malicious logic prior to and during the operation of system applications[8].

Post Office Protocol (POP)—an email storage protocol that allows email to be held at a server until a disconnected client can retrieve it. It is the current standard. [19].

Security kernel—the hardware, firmware, and software elements of a Trusted Computing Base that implement the reference monitor concept. It must mediate all accesses, be protected from modification, and be verifiable as correct [2].

Simple Mail Transfer Protocol (SMTP)—the current standard for email.

Subject—an active entity, generally in the form of a person, process, or device that causes information to flow among objects or changes the system state [2].

System-high mode—a mode in which all information on a system is held at the highest sensitivity level of any data on the system. All users must then be authorized to this highest level.

Trojan Horse—a computer program with an apparently or actually useful function that contains additional (hidden) functions that surreptitiously exploit the legitimate authorizations of the invoking process to the detriment of security [2].

Trusted Computing Base (TCB)—the totality of protection mechanisms within a computer system—including hardware, firmware and software—the combination of which is responsible for enforcing a security policy [2].

APPENDIX B: C SOURCE CODE

Makefile –IMap toolkit

```
# Filename:     Makefile --Top level
#
# Program:  IMAP Toolkit Makefile
#
# Author:   Mark Crispin
#           Networks and Distributed Computing
#           Computing & Communications
#           University of Washington
#           Administration Building, AG-44
#           Seattle, WA  98195
#           Internet: MRC@CAC.Washington.EDU
#
# Date:           7 December 1989
#
# Modified by:    Brad Eads
#           Naval Postgraduate School
#           Monterey CA 93940
#           17 Jan 1999
#
# Last Edited:    27 Feb 1999
#
# Copyright 1998 by the University of Washington
#
#  Permission to use, copy, modify, and distribute this software and its
# documentation for any purpose and without fee is hereby granted,
provided
# that the above copyright notice appears in all copies and that both
the
# above copyright notice and this permission notice appear in supporting
# documentation, and that the name of the University of Washington not
be
# used in advertising or publicity pertaining to distribution of the
software
# without specific, written prior permission.  This software is made
# available "as is", and
# THE UNIVERSITY OF WASHINGTON DISCLAIMS ALL WARRANTIES, EXPRESS OR
IMPLIED,
# WITH REGARD TO THIS SOFTWARE, INCLUDING WITHOUT LIMITATION ALL IMPLIED
# WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE,
AND IN
# NO EVENT SHALL THE UNIVERSITY OF WASHINGTON BE LIABLE FOR ANY SPECIAL,
# INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING
FROM
# LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, TORT
# (INCLUDING NEGLIGENCE) OR STRICT LIABILITY, ARISING OUT OF OR IN
CONNECTION
# WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.


# Normal command to build IMAP toolkit:
#  make <port> [EXTRAAUTHENTICATORS=xxx] [EXTRADRIVERS=xxx]
[PASSWDTYPE=xxx]
```

```
#
# For example:
#     make os4 EXTRAAUTHENTICATORS=gss
# builds for Digital Unix (OSF/1) version 4 with GSSAPI/Kerberos V
additional
# authentication.


# The following extra authenticators are defined:
# krb Kerberos IV (client-only)
# gss GSSAPI/Kerberos V

EXTRAAUTHENTICATORS=


# The following extra drivers are defined:
# mbox      if file "mbox" exists on the home directory, automatically
moves mail
#      from the spool directory to "mbox" and uses "mbox" as INBOX.

EXTRADRIVERS=mbox


# The following plaintext login types are defined:
# afs AFS authentication
# dce DCE authentication
# krb Kerberos IV (must also have krb as an extra authentication)
# nul no plaintext authentication (note: this will break some secure
#      authenticators -- don't use without checking first!!)
# std system standard

PASSWDTYPE=std

# Directory locations.  I didn't want to put this stuff here, but the
Pine
# guys insisted...

AFSDIR=/usr/afsws
GSSDIR=/usr/local


# Miscellaneous command line options passed down to the c-client
Makefile

EXTRACFLAGS=
EXTRALDFLAGS=

# Normal commands

CAT=cat
CD=cd
LN=ln -s
MAKE=make
MKDIR=mkdir
RM=rm -rf
SH=sh
SYSTEM=unix
```

```
TOOLS=tools
TOUCH=touch


# Primary build command

BUILDOPTIONS= EXTRACFLAGS="$(EXTRACFLAGS)"\
 EXTRALDFLAGS="$(EXTRALDFLAGS)"\
 EXTRADRIVERS="$(EXTRADRIVERS)"
EXTRAAUTHENTICATORS="$(EXTRAAUTHENTICATORS)"\
 PASSWDTYPE=$(PASSWDTYPE) AFSDIR=$(AFSDIR) GSSDIR=$(GSSDIR)
BUILD=$(MAKE) build $(BUILDOPTIONS) EXTRASPECIALS="$(EXTRASPECIALS)"


# Make the IMAP Toolkit

all:  c-client rebuild bundled


#  The following ports are defined.  These refer to the *standard*
compiler
# on the given system.  This means, for example, that the sol port is
for SUN's
# compiler and not for a non-standard compiler such as gcc.
#  If you are using gcc and it is not the standard compiler on your
system, try
# using an ANSI port that is clng bundled tools...
      $(CD) mtest;$(MAKE)
      $(CD) ipopd;$(MAKE)
      $(CD) imapd;$(MAKE)

clean:
      @echo Removing old processed sources and binaries...
      sh -c '$(RM) an ua OSTYPE c-client mtest imapd ipopd || true'
      $(CD) tools;$(MAKE) clean


# A monument to a hack of long ago and far away...
love:
      @echo not war?
```

Makefile –C client

```
# Filename:     Makefile --C client
#
# Program:      C client makefile
#
# Author:       Mark Crispin
#               Networks and Distributed Computing
#               Computing & Communications
#               University of Washington
#               Administration Building, AG-44
#               Seattle, WA  98195
#               Internet: MRC@CAC.Washington.EDU
#
# Date:         11 May 1989
#
# Modified by:  Brad Eads
#               Naval Postgraduate School
#               Monterey CA 93940
#               17 Jan 1999
#
# Last Edited:  27 Feb 1999
#
# Copyright 1998 by the University of Washington
#
#  Permission to use, copy, modify, and distribute this software and its
# documentation for any purpose and without fee is hereby granted, provided
# that the above copyright notice appears in all copies and that both the
# above copyright notice and this permission notice appear in supporting
# documentation, and that the name of the University of Washington not be
# used in advertising or publicity pertaining to distribution of the software
# without specific, written prior permission.  This software is made
# available "as is", and
# THE UNIVERSITY OF WASHINGTON DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,
# WITH REGARD TO THIS SOFTWARE, INCLUDING WITHOUT LIMITATION ALL IMPLIED
# WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND
IN
# NO EVENT SHALL THE UNIVERSITY OF WASHINGTON BE LIABLE FOR ANY SPECIAL,
# INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING
FROM
# LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, TORT
# (INCLUDING NEGLIGENCE) OR STRICT LIABILITY, ARISING OUT OF OR IN CONNECTION
# WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.


# Command line build parameters

EXTRAAUTHENTICATORS=
EXTRADRIVERS=mbox
PASSWDTYPE=std
```

```
# Extended flags needed for non-standard passwd types.  You may need to modify.

AFSDIR=/usr/afsws
AFSCFLAGS=-Dexit=afs_exit -D_exit=_afs_exit -I$(AFSDIR)/include
AFSLIB=$(AFSDIR)/lib
AFSLDFLAGS=-L$(AFSLIB)/afs -L$(AFSLIB)\
 -lkauth -lprot -lubik -lauth -lrxkad -lrx -llwp -ldes -lcom_err\
 $(AFSLIB)/afs/util.a -laudit -lsys
DCECFLAGS= -DDCE_MINIMAL -DPASSWD_OVERRIDE=\"/opt/pop3/passwd/passwd\"
DCELDFLAGS= -ldce


# Extended flags needed for additional authenticators.  You may need to modify.

KRBCFLAGS= -I/usr/include/kerberosIV
KRBLDFLAGS= -lkrb -ldes
GSSDIR=/usr/local
GSSCFLAGS= -I$(GSSDIR)/include
GSSLDFLAGS= -L$(GSSDIR)/lib -lgssapi_krb5 -lkrb5 -lcrypto -lcom_err


# Build parameters normally set by the individual port

CHECKPW=std
LOGINPW=std
SIGTYPE=bsd
ACTIVEFILE=/usr/lib/news/active
SPOOLDIR=/usr/spool
MAILSPOOL=$(SPOOLDIR)/mail
NEWSSPOOL=$(SPOOLDIR)/news
STDPROTO=unixproto
RSHPATH=/usr/ucb/rsh


# Commands possibly overriden by the individual port

ARRC=ar rc
CC=cc
LN=ln -s
RANLIB=ranlib


# Standard distribution build parameters

DEFAULTAUTHENTICATORS=log
DEFAULTDRIVERS=imap nntp pop3 mh mx mbx tenex mtx mmdf unix news phile


# Normally no need to change any of these
```

```
ARCHIVE=c-client.a
BINARIES=mail.o misc.o newsrc.o smanager.o osdep.o utf8.o siglocal.o \
 dummy.o pseudo.o netmsg.o flstring.o fdstring.o \
 rfc822.o nntp.o smtp.o imap4r1.o pop3.o \
 unix.o mbox.o mbx.o mmdf.o tenex.o mtx.o news.o phile.o mh.o mx.o
CFLAGS=$(BASECFLAGS) $(EXTRACFLAGS)
MAKE=make
MV=mv
RM=rm -rf
SH=sh


# Primary build command

BUILDOPTIONS= EXTRACFLAGS="$(EXTRACFLAGS)"\
 EXTRALDFLAGS="$(EXTRALDFLAGS)"\
 EXTRADRIVERS="$(EXTRADRIVERS)"
EXTRAAUTHENTICATORS="$(EXTRAAUTHENTICATORS)"\
 PASSWDTYPE=$(PASSWDTYPE) AFSDIR=$(AFSDIR) GSSDIR=$(GSSDIR)
BUILD=$(MAKE) build $(BUILDOPTIONS) $(EXTRASPECIALS)

# Here if no make argument established

missing: osdep.h
	$(MAKE) all CC=`cat CCTYPE` CFLAGS="`cat CFLAGS`"

osdep.h:
	@echo You must specify what type of system
	@false


# Current ports

a32:	# AIX 3.2 for RS/6000
	$(BUILD) OS=$@ SIGTYPE=psx \
	 SPOOLDIR=/var/spool \
	 RSHPATH=/usr/bin/rsh \
	 BASECFLAGS="-g -Dunix=1 -D_BSD -DNFSKLUDGE" \
	 BASELDFLAGS="-lbsd"

a41:	# AIX 4.1 for RS/6000
	$(BUILD) OS=$@ SIGTYPE=psx CHECKPW=a41 \
	 SPOOLDIR=/var/spool \
	 RSHPATH=/usr/bin/rsh \
	 BASECFLAGS="-g -Dunix=1 -D_BSD -DNFSKLUDGE" \
	 BASELDFLAGS="-lbsd -ls"

aix:	# AIX/370
	@echo You are building for AIX on an S/370 class machine
	@echo If you want AIX on an RS/6000 you need to use a32 or a41 instead!
	$(BUILD) OS=$@ \
         BASECFLAGS="-g -DNFSKLUDGE" \
         BASELDFLAGS="-lbsd"

aos:    # AOS for RT
        $(BUILD) OS=$@ \
         BASECFLAGS="-g -Dconst= -DNFSKLUDGE"

art:    # AIX 2.2.1 for RT
        $(BUILD) OS=$@ SIGTYPE=sv4 \
         SPOOLDIR=/var \
         ACTIVEFILE=/usr/local/news/control/active \
         RSHPATH=/bin/rsh \
         BASECFLAGS="-g -Dconst= -Dvoid=char" \
         RANLIB=true

asv:    # Altos SVR4
        $(BUILD) OS=$@ SIGTYPE=sv4 LOGINPW=old \
         ACTIVEFILE=/usr/spool/news/active \
         RSHPATH=/usr/bin/rcmd \
         BASECFLAGS="-Dconst= -DSIGSTOP=SIGKILL" \
         BASELDFLAGS="-lsocket -lrpc -lgen -lcrypt -lxenix" \
         RANLIB=true

aux:    # A/UX
        $(BUILD) OS=$@ \
         MAILSPOOL=/usr/mail \
         BASECFLAGS="-g -B/usr/lib/big/ -Dvoid=char -Dconst= -DNFSKLUDGE" \
         RANLIB=true ARRC="ar -rc"

bs3:    # BSD/i386 3.0 or higher
        $(BUILD) OS=bsi CHECKPW=bsi LOGINPW=bsi \
         MAILSPOOL=/var/mail NEWSSPOOL=/var/news/spool \
         ACTIVEFILE=/var/news/etc/active \
         RSHPATH=/usr/bin/rsh \
         BASECFLAGS="-g -O2 -pipe -DNFSKLUDGE" CC=shlicc

bsd:    # BSD UNIX
        $(BUILD) OS=$@ \
         BASECFLAGS="-g -Dconst= -DNFSKLUDGE"

bsf:    # FreeBSD
        $(BUILD) OS=bsi SIGTYPE=psx \
         SPOOLDIR=/var \
         ACTIVEFILE=/usr/local/news/lib/active \
         RSHPATH=/usr/bin/rsh \
         BASECFLAGS="-g -O -pipe -DNFSKLUDGE" \
         BASELDFLAGS="-lcrypt"

bsi:    # BSD/i386
        $(BUILD) OS=$@ \
         MAILSPOOL=/var/mail NEWSSPOOL=/var/news/spool \
         ACTIVEFILE=/var/news/etc/active \
         RSHPATH=/usr/bin/rsh \
         BASECFLAGS="-g -O2 -pipe -DNFSKLUDGE"

cvx:    # Convex
        $(BUILD) OS=$@ \
         BASECFLAGS="-O -ext -Dconst= -DNFDKLUDGE"

d-g:    # Data General DG/UX
        $(BUILD) OS=$@ SIGTYPE=sv4 \
         MAILSPOOL=/var/mail NEWSSPOOL=/var/spool/news \
         ACTIVEFILE=/local/news/active \
         RSHPATH=/usr/bin/remsh \
         BASECFLAGS="-g -Dconst= -DNFSKLUDGE" \
         BASELDFLAGS="-lnsl -lsocket" \
         RANLIB=true

d54:    # Data General DG/UX 5.4
        $(BUILD) OS=d-g SIGTYPE=sv4 \
         MAILSPOOL=/var/mail NEWSSPOOL=/var/spool/news \
         ACTIVEFILE=/local/news/active \
         RSHPATH=/usr/bin/remsh \
         BASECFLAGS="-g -Dconst= -DNFSKLUDGE" \
         RANLIB=true

dpx:    # Bull DPX/2
        $(BUILD) OS=sv4 SIGTYPE=sv4 CHECKPW=sv4 LOGINPW=sv4 \
         RSHPATH=/usr/bin/remsh \
         BASECFLAGS="-Dconst= -DSYSTEM5 -DSHORT_IDENT" \
         BASELDFLAGS="-linet" \
         RANLIB=true LN=ln

drs:    # ICL DRS/NX
        $(BUILD) OS=$@ SIGTYPE=sv4 CHECKPW=sv4 LOGINPW=sv4 \
         MAILSPOOL=/var/mail NEWSSPOOL=/var/spool/news \
         ACTIVEFILE=/var/lib/news/active \
         RSHPATH=/usr/bin/rsh \
         BASECFLAGS="-O -DNFSKLUDGE" \
         BASELDFLAGS="-lsocket -lgen" \
         RANLIB=true

dyn:    # Dynix
        $(BUILD) OS=$@ \
         BASECFLAGS="-g -Dconst= -DNFSKLUDGE"

epx:    # EP/IX
        $(BUILD) OS=sv4 SIGTYPE=sv4 CHECKPW=sv4 LOGINPW=sv4 \
         MAILSPOOL=/var/mail NEWSSPOOL=/var/spool/news \
         ACTIVEFILE=/usr/share/news/active \
         RSHPATH=/usr/net/rsh \
         BASECFLAGS="-g -systype svr4" \
         BASELDFLAGS="-lsocket -lnsl -lgen" \
         RANLIB=true

gas:    # GCC Altos SVR4
        $(BUILD) OS=asv SIGTYPE=sv4 LOGINPW=old \
         ACTIVEFILE=/usr/spool/news/active \
         RSHPATH=/usr/bin/rcmd \
         BASECFLAGS="-g -O -DALTOS_SYSTEM_V -DSIGSTOP=SIGKILL" \
         BASELDFLAGS="-lsocket -lrpc -lgen -lcrypt -lxenix" \
         RANLIB=true CC=gcc

gh9:    # GCC HP-UX9.x
        $(BUILD) OS=hpp SIGTYPE=psx \
         MAILSPOOL=/usr/mail \
         RSHPATH=/usr/bin/remsh \
         BASECFLAGS="-O2 -DNFSKLUDGE" \
         RANLIB=true CC=gcc

ghp:    # GCC HP-UX
        $(BUILD) OS=hpp SIGTYPE=psx \
         SPOOLDIR=/var \
         ACTIVEFILE=/var/news/active \
         RSHPATH=/usr/bin/remsh \
         BASECFLAGS="-O2 -DNFSKLUDGE" \
         RANLIB=true CC=gcc

gso:    # GCC Solaris
        $(BUILD) OS=sol SIGTYPE=psx CHECKPW=psx \
         MAILSPOOL=/var/mail NEWSSPOOL=/var/spool/news \
         ACTIVEFILE=/usr/share/news/active \
         RSHPATH=/usr/bin/rsh \
         BASECFLAGS="-g -O2 -DNFSKLUDGE" \
         BASELDFLAGS="-lsocket -lnsl -lgen" \
         RANLIB=true CC=gcc

gsu:    # GCC SUN-OS
        $(BUILD) OS=sun \
         BASECFLAGS="-O2 -DNFSKLUDGE" \
         BASELDFLAGS="-ldl" \
         CC=gcc

gul:    # GCC Ultrix
        $(BUILD) OS=ult SIGTYPE=psx CHECKPW=ult \
         BASECFLAGS="-g -O -DNFSKLUDGE" \
         BASELDFLAGS="-lauth" \
         CC=gcc

hpp:    # HP-UX 9.x
        $(BUILD) OS=$@ SIGTYPE=psx \
         MAILSPOOL=/usr/mail \
         RSHPATH=/usr/bin/remsh \
         BASECFLAGS="-g -Aa -D_HPUX_SOURCE -DNFSKLUDGE" \
         BASELDFLAGS="-lnet -lV3" \
         RANLIB=true

hpx:    # HP-UX 10.x
        $(BUILD) OS=hpp SIGTYPE=psx \
         SPOOLDIR=/var \
         ACTIVEFILE=/var/news/active \
         RSHPATH=/usr/bin/remsh \
         BASECFLAGS="-g -Ae -DNFSKLUDGE" \
         BASELDFLAGS="-lnet -lV3" \
         RANLIB=true

isc:    # Interactive
        $(BUILD) OS=$@ SIGTYPE=sv4 CHECKPW=sv4 LOGINPW=sv4 \
         MAILSPOOL=/var/mail NEWSSPOOL=/var/spool/news \
         ACTIVEFILE=/var/spool/news/active \
         BASECFLAGS="-Xp -D_SYSV3" \
         BASELDFLAGS="-linet -lnsl_s -lgen -lx -lsec -liberty" \
         RANLIB=true

lnx:    # Linux non-shadow passwords
        @echo You are building for traditional Linux *without* shadow
        @echo passwords and with the crypt function in the C library.
        @echo If your system has shadow passwords, or if crypt is not
        @echo in the C library, you must use slx, sl4, or sl5 instead!
        $(BUILD) OS=$@ SIGTYPE=psx \
         SPOOLDIR=/var/spool \
         RSHPATH=/usr/bin/rsh \
         BASECFLAGS="-g -O -DNFSKLUDGE"

lyn:    # LynxOS
        $(BUILD) OS=$@ \
         RSHPATH=/bin/rsh \
         BASECFLAGS="-g -O -pipe -DNFSKLUDGE" \
         BASELDFLAGS=-lbsd \
         CC=gcc

mct:    # MachTen - NFSKLUDGE doesn't work (at least not on 2.2)
        $(BUILD) OS=$@ \
         SPOOLDIR=/var/spool \
         BASECFLAGS="-g -O -pipe"

mnt:    # Mint
        $(BUILD) OS=$@ \
         BASECFLAGS="-g -O -Dunix=1 -D__atarist__ -D_POSIX_SOURCE -pipe" \
         RSHPATH=/usr/bin/rsh \
         RANLIB=true CC=gcc ARRC="gcc-ar rc"

neb:    # NetBSD
        $(BUILD) OS=bsi \
         SPOOLDIR=/var \
         ACTIVEFILE=/var/db/news/active \
         RSHPATH=/usr/bin/rsh \
         BASECFLAGS="-g -O -pipe -DNFSKLUDGE" \
         BASELDFLAGS="-lcrypt"

nxt:    # NEXTSTEP
        $(BUILD) OS=$@ \
         BASECFLAGS="-g -O -pipe -DNFSKLUDGE"

osf:    # OSF/1
        $(BUILD) OS=$@ SIGTYPE=psx \
         BASECFLAGS="-g3 -O2 -Olimit 1500 -DNFSKLUDGE"

# Note: sia_become_user() used by LOGINPW=os4 doesn't seem to work right.  The
# user doesn't get proper file access, and the process can't be killed.

os4:    # OSF/1 (Digital UNIX) 4
        $(BUILD) OS=$@ SIGTYPE=psx CHECKPW=os4 LOGINPW=sec \
         BASECFLAGS="-g3 -O2 -Olimit 1500 -DNFSKLUDGE" \
         BASELDFLAGS="-lsecurity -ldb -laud -lm"

ptx:    # PTX
        $(BUILD) OS=$@ SIGTYPE=psx CHECKPW=svo LOGINPW=sv4 \
         MAILSPOOL=/usr/mail \
         RSHPATH=/usr/bin/resh \
         BASECFLAGS="-Wc,-O3 -Wc,-seq -Dprivate=PRIVATE -DNFSKLUDGE" \
         BASELDFLAGS="-lseq -lsec -lsocket -linet -lnsl -lgen" \
         RANLIB=true

pyr:    # Pyramid
        $(BUILD) OS=$@ \
         BASECFLAGS="-g -Dconst= -DNFSKLUDGE"

qnx:    # QNX
        $(BUILD) OS=$@ CHECKPW=psx LOGINPW=old \
         RSHPATH=/usr/ucb/rsh \
         BASECFLAGS="-Otax -g -Dunix=1 -D_POSIX_SOURCE" \
         BASELDFLAGS="-g -N128k -llogin -lsocket -lunix"

s40:    # SUN-OS 4.0
        $(BUILD) OS=$@ \
         BASECFLAGS="-g -Dconst= -DNFSKLUDGE"

sc5:    # SCO Open Server 5.0
        $(BUILD) OS=$@ SIGTYPE=psx CHECKPW=sec LOGINPW=sec \
         STDPROTO=mmdfproto \
         SPOOLDIR=/var/spool \
         ACTIVEFILE=/var/lib/news/active \
         RSHPATH=/usr/bin/rcmd \
         BASECFLAGS="-O3 -dy -s" \
          BASELDFLAGS="-lsocket -lcrypt -lprot -lx -ltinfo -lm" \
          RANLIB=true

sco:      # Santa Cruz Operation
          $(BUILD) OS=$@ SIGTYPE=sv4 CHECKPW=sec LOGINPW=sec \
           STDPROTO=mmdfproto \
           RSHPATH=/usr/bin/rcmd \
           BASECFLAGS="-O3" \
           BASELDFLAGS="-lsocket -lprot -lcrypt_i -lx -los" \
           RANLIB=true LN=ln

# Note: setting _POSIX_SOURCE doesn't seem to build it as of SGI IRIX 5.3

sgi:      # Silicon Graphics
          $(BUILD) OS=$@ SIGTYPE=sv4 \
           MAILSPOOL=/usr/mail \
           RSHPATH=/usr/bsd/rsh \
           BASECFLAGS="-g -ansi -DNFSKLUDGE" \
           RANLIB=true

# Note: Mark Kaesling says that setluid() isn't in HP-UX with SecureWare.

shp:      # HP-UX with Trusted Computer Base
          $(BUILD) OS=$@ SIGTYPE=psx CHECKPW=sec LOGINPW=std \
           SPOOLDIR=/var \
           ACTIVEFILE=/var/news/active \
           RSHPATH=/usr/bin/remsh \
           BASECFLAGS="-g -Ae -DNFSKLUDGE" \
           BASELDFLAGS="-lnet -lV3 -lsec" \
           RANLIB=true

slx:      # Secure Linux
          @echo You are building for libc6/glibc versions of Secure Linux
          @echo If you want libc5 versions you must use sl5 instead!
          @echo If you want libc4 versions you must use sl4 instead!
          $(BUILD) OS=$@ SIGTYPE=psx CHECKPW=psx \
           SPOOLDIR=/var/spool \
           RSHPATH=/usr/bin/rsh \
           BASECFLAGS="-g -fno-omit-frame-pointer -O6 -DNFSKLUDGE" \
           BASELDFLAGS="-s -lcrypt"

sl4:      # Secure Linux using libc4
          @echo You are building for libc4 versions of Secure Linux
          @echo If you want libc6/glibc versions you must use slx instead!
          @echo If you want libc5 versions you must use sl5 instead!
          $(BUILD) OS=slx SIGTYPE=psx CHECKPW=psx \
           SPOOLDIR=/var/spool \
           RSHPATH=/usr/bin/rsh \
           BASECFLAGS="-g -fno-omit-frame-pointer -O6 -DNFSKLUDGE" \
           BASELDFLAGS="-s -lshadow"
```

```
sl5:    # Secure Linux using libc5
        @echo You are building for libc5 versions of Secure Linux
        @echo If you want libc6/glibc versions you must use slx instead!
        @echo If you want libc4 versions you must use sl4 instead!
        $(BUILD) OS=slx SIGTYPE=psx CHECKPW=psx \
         SPOOLDIR=/var/spool \
         RSHPATH=/usr/bin/rsh \
         BASECFLAGS="-g -fno-omit-frame-pointer -O6 -DNFSKLUDGE"

snx:    # Siemens Nixdorf SINIX and Reliant UNIX
        $(BUILD) OS=sv4 SIGTYPE=psx CHECKPW=sv4 \
         MAILSPOOL=/var/mail NEWSSPOOL=/var/spool/news \
         ACTIVEFILE=/usr/share/news/active \
         RSHPATH=/usr/bin/rsh \
         BASECFLAGS="-g -D_SYS_CLOCK_H -Dconst=" \
         BASELDFLAGS="-lsocket -lnsl -lgen" \
         RANLIB=true

# Note: It is a long and disgusting story about why cc is set to ucbcc.  You
# need to invoke the C compiler so that it links with the SVR4 libraries and
# not the BSD libraries, otherwise readdir() will return the wrong information.
# Of all the names in the most common path, ucbcc is the only name to be found
# (on /usr/ccs/bin) that points to a suitable compiler.  cc is likely to be
# /usr/ucb/cc which is absolutely not the compiler that you want.  The real
# SVR4 cc is probably something like /opt/SUNWspro/bin/cc which is rarely in
# anyone's path.
#
# ucbcc is probably a link to acc, e.g. /opt/SUNWspro/SC4.0/bin/acc, and is
# the UCB C compiler using the SVR4 libraries.
#
# If ucbcc isn't on your system, then punt on the SUN C compiler and use gcc
# instead (the gso port instead of the sol port).
#
# If, in spite of all the above warnings, you choose to change "ucbcc" to "cc",
# you will probably find that the -O2 needs to be changed to -O.  If you don't
# get any error messages with -O2, that's a pretty good indicator that you
# goofed and are running the compiler that will link with the BSD libraries.
#
# To recap:
# 1) The sol port is designed to be built using the UCB compiler using the
#    SVR4 libraries.  This compiler is "ucbcc", which is lunk to acc.  You
#    use -O2 as one of the CFLAGS.
# 2) If you build the sol port with the UCB compiler using the BSD libraries,
#    you will get no error messages but you will get bad binaries (the most
#    obvious symptom is dropping the first two characters return filenames
#    from the imapd LIST command.  This compiler also uses -O2, and is very
#    often what the user gets from "cc".  BEWARE!!!
# 3) If you build the sol port with the real SVR4 compiler, which is often
#    hidden away or unavailable on many systems, then you will get errors
#    from -O2 and you need to change that to -O.  But you will get a good
#    binary.  However, you should try it with -O2 first, to make sure that
#   you got this compiler and not the UCB compiler using BSD libraries.

sol:      # Solaris
          $(BUILD) OS=$@ SIGTYPE=psx CHECKPW=psx \
           MAILSPOOL=/var/mail NEWSSPOOL=/var/spool/news \
           ACTIVEFILE=/usr/share/news/active \
           RSHPATH=/usr/bin/rsh \
           BASECFLAGS="-g -O2 -DNFSKLUDGE" \
           BASELDFLAGS="-lsocket -lnsl -lgen" \
           RANLIB=true CC=ucbcc

sos:      # Secure OSF/1
          $(BUILD) OS=$@ SIGTYPE=psx CHECKPW=sec LOGINPW=sec \
           BASECFLAGS="-g3 -O2 -Olimit 1500 -DNFSKLUDGE" \
           BASELDFLAGS="-lsecurity -laud"

ssn:      # Secure SUN-OS
          $(BUILD) OS=sun CHECKPW=ssn \
           BASECFLAGS="-g -Dconst= -DNFSKLUDGE" \
           BASELDFLAGS="-ldl"

sun:      # SUN-OS
          $(BUILD) OS=$@ \
           BASECFLAGS="-g -Dconst= -DNFSKLUDGE" \
           BASELDFLAGS="-ldl"

sv2:      # SVR2
          @echo You are being *very* optimistic!
          $(BUILD) OS=$@ SIGTYPE=sv4 LOGINPW=old \
           MAILSPOOL=/usr/mail \
           RSHPATH=/usr/bin/remsh \
           BASECFLAGS="-Dconst= -DSYSTEM5 -DSHORT_IDENT -I/usr/ethernet/include" \
           BASELDFLAGS="-lnet" \
           RANLIB=true LN=ln

sv4:      # SVR4
          $(BUILD) OS=$@ SIGTYPE=sv4 CHECKPW=sv4 LOGINPW=sv4 \
           MAILSPOOL=/var/mail NEWSSPOOL=/var/spool/news \
           ACTIVEFILE=/usr/share/news/active \
           RSHPATH=/usr/bin/resh \
           BASECFLAGS="-g -Dconst=" \
           BASELDFLAGS="-lsocket -lnsl -lgen" \
           RANLIB=true

ult:      # Ultrix
          $(BUILD) OS=$@ SIGTYPE=psx CHECKPW=ult \
           BASECFLAGS="-g3 -O2 -Olimit 1500 -Dconst= -DNFSKLUDGE" \
           BASELDFLAGS="-lauth"

uw2:      # UnixWare SVR4.2
          $(BUILD) OS=sv4 SIGTYPE=sv4 CHECKPW=sv4 \
             MAILSPOOL=/var/mail NEWSSPOOL=/var/spool/news \
             ACTIVEFILE=/var/news/lib/active \
             RSHPATH=/usr/bin/rsh \
             BASECFLAGS="-g" \
             BASELDFLAGS="-lsocket -lnsl -lgen" \
             RANLIB=true

vul:        # VAX Ultrix
            $(BUILD) OS=ult SIGTYPE=psx CHECKPW=ult \
             BASECFLAGS="-O2 -Dconst= -DNFSKLUDGE" \
             BASELDFLAGS="-lauth"

vu2:        # VAX Ultrix 2.3, etc.
            $(BUILD) OS=$@ \
             BASECFLAGS="-O2 -Dconst= -Dvoid=char -DNFSKLUDGE"

#*****************************************************************
#                    Added Code
#*****************************************************************
xts:        # XTS-300
            @echo Now compiling for XTS-300!!
            $(BUILD) OS=$@ SIGTYPE=sv4 LOGINPW=sv4 \
             MAILSPOOL=/usr/mail \
             RSHPATH=/bin/rsh \
             BASECFLAGS="-g -Dconst= -DSYSTEM5 -DSHORT_IDENT -I/usr/ethernet/include" \
             BASELDFLAGS="-lcass -lsocket -lc -lgen" \
             RANLIB=true
        LN=ln

# Build it!

build:      clean once $(ARCHIVE)

all:        $(ARCHIVE)

$(ARCHIVE): $(BINARIES)
            sh -c '$(RM) $(ARCHIVE) || true'
            @cat ARCHIVE
            @$(SH) ARCHIVE

# Cleanup

clean:
            sh -c '$(RM) auths.c flockbsd.c linkage.[ch] siglocal.c osdep*.[ch] *.o ARCHIVE *FLAGS
*TYPE $(ARCHIVE) || true'


# Dependencies

dummy.o: mail.h misc.h osdep.h dummy.h
fdstring.o: mail.h misc.h osdep.h fdstring.h
```

```
flstring.o: mail.h misc.h osdep.h flstring.h
imap4r1.o: mail.h misc.h osdep.h imap4r1.h rfc822.h
mail.o: mail.h misc.h osdep.h rfc822.h linkage.h
mbox.o: mail.h misc.h osdep.h mbox.h unix.h
mbx.o: mail.h misc.h osdep.h mbx.h dummy.h
mh.o: mail.h misc.h osdep.h mh.h dummy.h
mx.o: mail.h misc.h osdep.h mx.h dummy.h
misc.o: mail.h misc.h osdep.h
mmdf.o: mail.h misc.h osdep.h mmdf.h pseudo.h dummy.h
mtx.o: mail.h misc.h osdep.h mtx.h dummy.h
netmsg.o: mail.h misc.h osdep.h netmsg.h
news.o: mail.h misc.h osdep.h news.h
newsrc.o: mail.h misc.h osdep.h newsrc.h
nntp.o: mail.h misc.h osdep.h netmsg.h smtp.h nntp.h rfc822.h
phile.o: mail.h misc.h osdep.h phile.h rfc822.h dummy.h
pseudo.o: pseudo.h
pop3.o: mail.h misc.h osdep.h pop3.h
smanager.o: mail.h misc.h osdep.h
smtp.o: mail.h misc.h osdep.h smtp.h rfc822.h
rfc822.o: mail.h misc.h osdep.h rfc822.h
tenex.o: mail.h misc.h osdep.h tenex.h dummy.h
unix.o: mail.h misc.h osdep.h unix.h pseudo.h dummy.h
utf8.o: mail.h misc.h osdep.h utf8.h


# OS-dependent

osdep.o:mail.h misc.h env.h fs.h ftl.h nl.h tcp.h \
        osdep.h env_unix.h tcp_unix.h \
        osdep.c env_unix.c fs_unix.c ftl_unix.c nl_unix.c tcp_unix.c \
        auths.c flock.c flockbsd.c flcksafe.c fsync.c gethstid.c \
        gr_wait.c gr_wait4.c gr_waitp.c \
        auth_gss.c auth_krb.c auth_log.c \
        scandir.c setpgrp.c strerror.c truncate.c write.c \
        memmove.c memmove2.c memset.c \
        tz_bsd.c tz_nul.c tz_sv4.c \
        write.c \
        strerror.c strpbrk.c strstr.c strtok.c strtoul.c \
        OSCFLAGS
        $(CC) $(CFLAGS) `cat OSCFLAGS` -c osdep.c

osdep.c: osdepbas.c osdepckp.c osdeplog.c
        sh -c '$(RM) osdep.c || true'
        cat osdepbas.c osdepckp.c osdeplog.c > osdep.c

flockbsd.c:     # cretin Linux

siglocal.c:     # cretin Linux


# Once-only environment setup
```

```
once:    onceenv ckp$(PASSWDTYPE) osdep.c $(EXTRAAUTHENTICATORS)

onceenv:
         @echo Once-only environment setup...
         ./drivers $(EXTRADRIVERS) $(DEFAULTDRIVERS) dummy
         ./mkauths $(EXTRAAUTHENTICATORS) $(DEFAULTAUTHENTICATORS)
         echo $(CC) > CCTYPE
         echo $(CFLAGS) > CFLAGS
         echo -DSTDPROTO=$(STDPROTO) -DMAILSPOOL=\"$(MAILSPOOL)\" \
          -DANONYMOUSHOME=\"$(MAILSPOOL)/anonymous\" \
          -DACTIVEFILE=\"$(ACTIVEFILE)\" -DNEWSSPOOL=\"$(NEWSSPOOL)\" \
          -DRSHPATH=\"$(RSHPATH)\" > OSCFLAGS
         echo $(BASELDFLAGS) $(EXTRALDFLAGS) > LDFLAGS
         echo "$(ARRC) $(ARCHIVE) $(BINARIES);$(RANLIB) $(ARCHIVE)" >> ARCHIVE
         $(LN) os_$(OS).h osdep.h
         $(LN) os_$(OS).c osdepbas.c
         $(LN) log_$(LOGINPW).c osdeplog.c
         $(LN) sig_$(SIGTYPE).c siglocal.c
         sh -c '(test -f /usr/include/sys/statvfs.h -a $(OS) != sc5 -a $(OS) != sco) && $(LN) flocksun.c
flockbsd.c || $(LN) flocksv4.c flockbsd.c'

# Password checkers

ckpafs:  # AFS
         echo $(AFSCFLAGS) >> OSCFLAGS
         echo $(AFSLDFLAGS) >> LDFLAGS
         $(LN) ckp_afs.c osdepckp.c

ckpdce:  # DCE
         echo $(DCECFLAGS) >> OSCFLAGS
         echo $(DCELDFLAGS) >> LDFLAGS
         $(LN) ckp_dce.c osdepckp.c

ckpkrb:  # Kerberos IV (must have krb EXTRAAUTHENTICATOR as well)
         $(LN) ckp_krb.c osdepckp.c

ckpstd:  # Port standard
         $(LN) ckp_$(CHECKPW).c osdepckp.c


# Authenticators

krb:     # Kerberos IV flags
         echo $(KRBCFLAGS) >> OSCFLAGS
         echo $(KRBLDFLAGS) >> LDFLAGS

gss:     # GSSAPI Kerberos V flags
         echo $(GSSCFLAGS) >> OSCFLAGS
         echo $(GSSLDFLAGS) >> LDFLAGS
```

```
# A monument to a hack of long ago and far away...

love:
	@echo not war?
```

UNIX_ENV.C

```
/*
 * Program:       UNIX environment routines
 *
 * Author:        Mark Crispin
 *                Networks and Distributed Computing
 *                Computing & Communications
 *                University of Washington
 *                Administration Building, AG-44
 *                Seattle, WA  98195
 *                Internet: MRC@CAC.Washington.EDU
 *
 * Date:  1 August 1988
 *
 * Modified by: Brad Eads
 *                Naval Postgraduate School
 *                Monterey CA 93940
 *                12 Feb 1999
 *
 * Last Edited:   28 Feb 1999
 *
 * Copyright 1998 by the University of Washington
 *
 *  Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose and without fee is hereby granted, provided
 * that the above copyright notice appears in all copies and that both the
 * above copyright notice and this permission notice appear in supporting
 * documentation, and that the name of the University of Washington not be
 * used in advertising or publicity pertaining to distribution of the software
 * without specific, written prior permission.  This software is made available
 * "as is", and
 * THE UNIVERSITY OF WASHINGTON DISCLAIMS ALL WARRANTIES, EXPRESS OR
IMPLIED,
 * WITH REGARD TO THIS SOFTWARE, INCLUDING WITHOUT LIMITATION ALL IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND
IN
 * NO EVENT SHALL THE UNIVERSITY OF WASHINGTON BE LIABLE FOR ANY SPECIAL,
 * INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER
RESULTING FROM
 * LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, TORT
 * (INCLUDING NEGLIGENCE) OR STRICT LIABILITY, ARISING OUT OF OR IN CONNECTION
 * WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

#include <signal.h>
#include <sys/wait.h>
#include "write.c"                /* include safe writing routines */

/* Get all authenticators */
```

```
#include "auths.c"

/* c-client environment parameters */

static char *anonymous_user = "nobody";
static char *unlogged_user = "root";

static char *myUserName = NIL;  /* user name */
static char *myHomeDir = NIL;   /* home directory name */
static char *myLocalHost = NIL; /* local host name */
static char *myNewsrc = NIL;    /* newsrc file name */
static char *sysInbox = NIL;    /* system inbox name */
static char *newsActive = NIL;  /* news active file */
static char *newsSpool = NIL;   /* news spool */
                                /* anonymous home directory */
static char *anonymousHome = NIL;
static char *ftpHome = NIL;     /* ftp export home directory */
static char *publicHome = NIL;  /* public home directory */
static char *sharedHome = NIL;  /* shared home directory */
static char *blackBoxDir = NIL; /* black box directory name */
                                /* black box default home directory */
static char *blackBoxDefaultHome = NIL;
static short anonymous = NIL;   /* is anonymous */
static short blackBox = NIL;    /* is a black box */
static short has_no_life = NIL; /* is a cretin with no life */
static long list_max_level = 20;/* maximum level of list recursion */
                                /* default file protection */
static long mbx_protection = 0600;
                                /* default directory protection */
static long dir_protection = 0700;
                                /* default lock file protection */
static long lock_protection = 0666;
static long disableFcntlLock =  /* flock() emulator is a no-op */
#ifdef SVR4_DISABLE_FLOCK
 T
#else
 NIL
#endif
 ;
static long lockEaccesError =   /* warning on EACCES errors on .lock files */
#ifdef IGNORE_LOCK_EACCES_ERRORS
 NIL
#else
 T
#endif
 ;
                                /* default prototypes */
static MAILSTREAM *createProto = NIL;
static MAILSTREAM *appendProto = NIL;
                                /* default user flags */
static char *userFlags[NUSERFLAGS] = {NIL};
static NAMESPACE *nslist[3];      /* namespace list */
static int logtry = 3;            /* number of login tries */

/* UNIX namespaces */

                                  /* personal mh namespace */
static NAMESPACE nsmhf = {"#mh/",'/',NIL,NIL};
static NAMESPACE nsmh = {"#mhinbox",NIL,NIL,&nsmhf};
                                  /* home namespace */
static NAMESPACE nshome = {"",'/',NIL,&nsmh};
                                  /* UNIX other user namespace */
static NAMESPACE nsunixother = {"~",'/',NIL,NIL};
                                  /* black box other user namespace */
static NAMESPACE nsblackother = {"/",'/',NIL,NIL};
                                  /* public (anonymous OK) namespace */
static NAMESPACE nspublic = {"#public/",'/',NIL,NIL};
                                  /* netnews namespace */
static NAMESPACE nsnews = {"#news.",'.',NIL,&nspublic};
                                  /* FTP export namespace */
static NAMESPACE nsftp = {"#ftp/",'/',NIL,&nsnews};
                                  /* shared (no anonymous) namespace */
static NAMESPACE nsshared = {"#shared/",'/',NIL,&nsftp};
```

```
/* Environment manipulate parameters
 * Accepts: function code
 *           function-dependent value
 * Returns: function-dependent return value
 */

void *env_parameters (long function,void *value)
{
 switch ((int) function) {
 case SET_NAMESPACE:
  fatal ("SET_NAMESPACE not permitted");
 case GET_NAMESPACE:
  value = (void *) nslist;
  break;
 case SET_USERNAME:
  if (myUserName) fs_give ((void **) &myUserName);
  myUserName = cpystr ((char *) value);
  break;
 case GET_USERNAME:
  value = (void *) myUserName;
  break;
 case SET_HOMEDIR:
  if (myHomeDir) fs_give ((void **) &myHomeDir);
  myHomeDir = cpystr ((char *) value);
  break;
 case GET_HOMEDIR:
  value = (void *) myHomeDir;
  break;
 case SET_LOCALHOST:
  if (myLocalHost) fs_give ((void **) &myLocalHost);
  myLocalHost = cpystr ((char *) value);
  break;
 case GET_LOCALHOST:
  value = (void *) myLocalHost;
  break;
 case SET_NEWSRC:
  if (myNewsrc) fs_give ((void **) &myNewsrc);
  myNewsrc = cpystr ((char *) value);
  break;
 case GET_NEWSRC:
  value = (void *) myNewsrc;
  break;
 case SET_NEWSACTIVE:
  if (newsActive) fs_give ((void **) &newsActive);
  newsActive = cpystr ((char *) value);
  break;
 case GET_NEWSACTIVE:
  value = (void *) newsActive;
  break;
```

```
case SET_NEWSSPOOL:
 if (newsSpool) fs_give ((void **) &newsSpool);
 newsSpool = cpystr ((char *) value);
 break;
case GET_NEWSSPOOL:
 value = (void *) newsSpool;
 break;
case SET_ANONYMOUSHOME:
 if (anonymousHome) fs_give ((void **) &anonymousHome);
 anonymousHome = cpystr ((char *) value);
 break;
case GET_ANONYMOUSHOME:
 value = (void *) anonymousHome;
 break;
case SET_FTPHOME:
 if (ftpHome) fs_give ((void **) &ftpHome);
 ftpHome = cpystr ((char *) value);
 break;
case GET_FTPHOME:
 value = (void *) ftpHome;
 break;
case SET_PUBLICHOME:
 if (publicHome) fs_give ((void **) &publicHome);
 publicHome = cpystr ((char *) value);
 break;
case GET_PUBLICHOME:
 value = (void *) publicHome;
 break;
case SET_SHAREDHOME:
 if (sharedHome) fs_give ((void **) &sharedHome);
 sharedHome = cpystr ((char *) value);
 break;
case GET_SHAREDHOME:
 value = (void *) sharedHome;
 break;
case SET_SYSINBOX:
 if (sysInbox) fs_give ((void **) &sysInbox);
 sysInbox = cpystr ((char *) value);
 break;
case GET_SYSINBOX:
 value = (void *) sysInbox;
 break;
case SET_LISTMAXLEVEL:
 list_max_level = (long) value;
 break;
case GET_LISTMAXLEVEL:
 value = (void *) list_max_level;
 break;
```

```
    case SET_MBXPROTECTION:
      mbx_protection = (long) value;
      break;
    case GET_MBXPROTECTION:
      value = (void *) mbx_protection;
      break;
    case SET_DIRPROTECTION:
      dir_protection = (long) value;
      break;
    case GET_DIRPROTECTION:
      value = (void *) dir_protection;
      break;
    case SET_LOCKPROTECTION:
      lock_protection = (long) value;
      break;
    case GET_LOCKPROTECTION:
      value = (void *) lock_protection;
      break;
    case SET_DISABLEFCNTLLOCK:
      disableFcntlLock = (long) value;
      break;
    case GET_DISABLEFCNTLLOCK:
      value = (void *) disableFcntlLock;
      break;
    case SET_LOCKEACCESERROR:
      lockEaccesError = (long) value;
      break;
    case GET_LOCKEACCESERROR:
      value = (void *) lockEaccesError;
      break;
    case SET_USERHASNOLIFE:
      has_no_life = value ? T : NIL;
      break;
    case GET_USERHASNOLIFE:
      value = (void *) (has_no_life ? T : NIL);
      break;
    default:
      value = NIL;              /* error case */
      break;
    }
    return value;
  }
```

```
/* Write current time
 * Accepts: destination string
 *          optional format of day-of-week prefix
 *          format of date and time
 *          flag whether to append symbolic timezone
 */

static void do_date (char *date,char *prefix,char *fmt,int suffix)
{
  time_t tn = time (0);
  struct tm *t = gmtime (&tn);
  int zone = t->tm_hour * 60 + t->tm_min;
  int julian = t->tm_yday;
  t = localtime (&tn);                /* get local time now */
                                      /* minus UTC minutes since midnight */
  zone = t->tm_hour * 60 + t->tm_min - zone;
  /* julian can be one of:
   *  36x  local time is December 31, UTC is January 1, offset -24 hours
   *    1  local time is 1 day ahead of UTC, offset +24 hours
   *    0  local time is same day as UTC, no offset
   *   -1  local time is 1 day behind UTC, offset -24 hours
   * -36x  local time is January 1, UTC is December 31, offset +24 hours
   */
  if (julian = t->tm_yday -julian)
    zone += ((julian < 0) == (abs (julian) == 1)) ? -24*60 : 24*60;
  if (prefix) {                       /* want day of week? */
    sprintf (date,prefix,days[t->tm_wday]);
    date += strlen (date);    /* make next sprintf append */
  }
                                      /* output the date */
  sprintf (date,fmt,t->tm_mday,months[t->tm_mon],t->tm_year+1900,
           t->tm_hour,t->tm_min,t->tm_sec,zone/60,abs (zone) % 60);
                                      /* append timezone suffix if desired */
  if (suffix) rfc822_timezone (date,(void *) t);
}


/* Write current time in RFC 822 format
 * Accepts: destination string
 */

void rfc822_date (char *date)
{
  do_date (date,"%s, ","%d %s %d %02d:%02d:%02d %+03d%02d",T);
}


/* Write current time in internal format
 * Accepts: destination string
 */
```

```
void internal_date (char *date)
{
 do_date (date,NIL,"%02d-%s-%d %02d:%02d:%02d %+03d%02d",NIL);
}
```

```
/* Set server traps
 * Accepts: clock interrupt handler
 *           kiss-of-death interrupt handler
 *           hangup interrupt handler
 *           termination interrupt handler
 */

void server_traps (void *clkint,void *kodint,void *hupint,void *trmint)
{
  arm_signal (SIGALRM,clkint);    /* prepare for clock interrupt */
  arm_signal (SIGUSR2,kodint);    /* prepare for Kiss Of Death */
  arm_signal (SIGHUP,hupint);     /* prepare for hangup */
  arm_signal (SIGTERM,trmint);    /* prepare for termination */
}


/* Wait for stdin input
 * Accepts: timeout in seconds
 * Returns: T if have input on stdin, else NIL
 */

long server_input_wait (long seconds)
{
  fd_set rfd;
  struct timeval tmo;
  FD_ZERO (&rfd);
  FD_SET (0,&rfd);
  tmo.tv_sec = seconds; tmo.tv_usec = 0;
  return select (1,&rfd,0,0,&tmo) ? LONGT : NIL;
}
```

```
/* Server log in
 * Accepts: user name string
 *          password string
 *          argument count
 *          argument vector
 * Returns: T if password validated, NIL otherwise
 */

long server_login (char *user,char *pwd,int argc,char *argv[])
{
 char *s,usr[MAILTMPLEN];
 struct passwd *pw;
                                /* cretins still haven't given up */
 if (strlen (user) >= MAILTMPLEN)
  syslog (LOG_ALERT|LOG_AUTH,"System break-in attempt, host=%.80s",
          tcp_clienthost ());
                                /* validate with case-independence */
 else if ((logtry > 0) && ((pw = getpwnam (strcpy (usr,user))) ||
                          (pw = getpwnam (lcase (usr)))) &&
         ((pw = checkpw (pw,pwd,argc,argv)) ||
          ((*pwd == ' ') && (pw = getpwnam (usr)) &&
           (pw = checkpw (pw,pwd + 1,argc,argv)))))
  return pw_login (pw,pw->pw_name,pw->pw_dir,argc,argv);
 s = (logtry-- > 0) ? "Login failure" : "Excessive login attempts";
                                /* note the failure in the syslog */
 syslog (LOG_INFO,"%s user=%.80s host=%.80s",s,user,tcp_clienthost ());
 sleep (3);                     /* slow down possible cracker */
 return NIL;
}
```

```
/* Authenticated server log in
 * Accepts: user name string
 *           argument count
 *           argument vector
 * Returns: T if password validated, NIL otherwise
 */

long authserver_login (char *user,int argc,char *argv[])
{
  struct passwd *pw = getpwnam (user);
  return pw ? pw_login (pw,pw->pw_name,pw->pw_dir,argc,argv) : NIL;
}


/* Log in as anonymous daemon
 * Accepts: argument count
 *           argument vector
 * Returns: T if successful, NIL if error
 */

long anonymous_login (int argc,char *argv[])
{
  struct passwd *pw = getpwnam (anonymous_user);
                                      /* log in Mr. A. N. Onymous */
  return pw ? pw_login (pw,NIL,NIL,argc,argv) : NIL;
}


/* Finish log in and environment initialization
 * Accepts: passwd struct for loginpw()
 *           user name (NIL for anonymous)
 *           home directory (NIL for anonymous)
 *           argument count
 *           argument vector
 * Returns: T if successful, NIL if error
 */

long pw_login (struct passwd *pw,char *user,char *home,int argc,char *argv[])
{
  if (pw->pw_uid && ((pw->pw_uid == geteuid ()) || loginpw (pw,argc,argv)) &&
      env_init (user,home)) {
    chdir (myhomedir ());   /* placate those who would be upset */
    return LONGT;
  }
  return NIL;
}
```

```
/* Initialize environment
 * Accepts: user name
 *          home directory name
 * Returns: T, always
 */

long env_init (char *user,char *home)
{
 extern MAILSTREAM STDPROTO;
 struct passwd *pw;
 struct stat sbuf;
 char *s,tmp[MAILTMPLEN];
/****************************
        Added Code
 ****************************/
  access_ma *user_level;  /*Mandatory access type holds MA parameters*/
  char *null_path = NIL;  /*Used to supply NIL to function getlevel()*/
  char slash[] = "/";                /*Used to supply "/" for directory path*/
  char *add_dir = '\0';     /*Used to store additional path name*/
  char secret[] = "secret";
  char conf[]  = "conf";
  char unclass[]= "unclass";
  char tmp2[MAILTMPLEN];
  getlevel(null_path, user_level);
  if(user_level->security_level==2) /* Are we at the secret level?*/
  {
    add_dir = secret;  /*Add the secret path*/
  }
  else if(user_level->security_level==1) /* Are we at the confidential level?*/
  {
    add_dir = conf;   /*Add the conf path*/
  }
  else  /* Everything else is unclassified*/
  {
    add_dir = unclass; /*Add the unclass path*/
  }

  int ix=0;          /* counter for loop*/
  while (*home && (home[ix] != '/'))         /*Loop until the slash*/
    {
    tmp2[ix] = home[ix];
    ix++;
    }              /*tmp should hold the first path in the directory*/
  strcat (tmp2, slash);  /* Add / to path*/
  strcat (tmp2, add_dir);  /* Add additional dir to path*/
  strcat (tmp2, slash);  /* Add / to path*/
  strcat (tmp2,user);/* Complete the path with the user's directory*/
/***************************************************************************/

 if (myUserName) fatal ("env_init called twice!");
```

```
                                        /* myUserName must be set before dorc() call */
myUserName = cpystr (user ? user : anonymous_user);
                                        /* do systemwide configuration */
dorc ("/etc/c-client.cf",NIL);
if (!anonymousHome) anonymousHome = cpystr (ANONYMOUSHOME);
if (user) {
        /* remember user name and home directory */
  if (blackBoxDir) {                    /* build black box directory name */

    sprintf (tmp,"%s/%s",blackBoxDir,myUserName);
                                        /* if black box if exists and directory */
    if (s = (!stat (tmp,&sbuf) && (sbuf.st_mode & S_IFDIR)) ?
          tmp : blackBoxDefaultHome) {
        sprintf (tmp,"%s/INBOX",myHomeDir = cpystr (s));
        sysInbox = cpystr (tmp);/* set black box values in their place */
        blackBox = T;
    }
  }
  if (blackBox)                 /* black box? */
    nslist[0] = &nshome,nslist[1] = &nsblackother,nslist[2] = &nsshared;
  else {                        /* not a black box */
    nslist[0] = &nshome,nslist[1] = &nsunixother,nslist[2] = &nsshared;
    myHomeDir = cpystr (tmp2);/* use real home directory */
                                        /* make sure user rc files don't try this */
    blackBoxDir = blackBoxDefaultHome = "";
  }
}
else {                          /* anonymous user */
  nslist[0] = nslist[1] = NIL,nslist[2] = &nsftp;
  sprintf (tmp,"%s/INBOX",myHomeDir = cpystr (anonymousHome));
  sysInbox = cpystr (tmp);              /* make system INBOX */
  anonymous = T;                        /* flag as anonymous */
                                        /* make sure an error message happens */
  if (!blackBoxDir) blackBoxDir = blackBoxDefaultHome = anonymousHome;
}
dorc (strcat (strcpy (tmp,myHomeDir),"/.mminit"),T);
dorc (strcat (strcpy (tmp,myHomeDir),"/.imaprc"),NIL);
if (!myLocalHost) mylocalhost ();
if (!myNewsrc) myNewsrc = cpystr(strcat (strcpy (tmp,myHomeDir),"/.newsrc"));
if (!newsActive) newsActive = cpystr (ACTIVEFILE);
if (!newsSpool) newsSpool = cpystr (NEWSSPOOL);
if (!ftpHome && (pw = getpwnam ("ftp"))) ftpHome = cpystr (pw->pw_dir);
if (!publicHome && (pw = getpwnam ("imappublic")))
  publicHome = cpystr (pw->pw_dir);
if (!anonymous && !sharedHome && (pw = getpwnam ("imapshared")))
  sharedHome = cpystr (pw->pw_dir);
                                        /* force default prototype to be set */
if (!createProto) createProto = &STDPROTO;
if (!appendProto) appendProto = &STDPROTO;
                                        /* re-do open action to get flags */
(*createProto->dtb->open) (NIL);
```

```
  endpwent ();                          /* close pw database */
  return T;
 }

/* Return my user name
 * Accepts: pointer to optional flags
 * Returns: my user name
 */

char *myusername_full (unsigned long *flags)
{
  char *ret = unlogged_user;
  if (!myUserName) {                    /* get user name if don't have it yet */
    struct passwd *pw;
    unsigned long euid = geteuid ();
    char *s = (char *) getlogin ();
                                        /* look up getlogin() user name or EUID */
    if (!((s && *s && (pw = getpwnam (s)) && (pw->pw_uid == euid)) ||
            (pw = getpwuid (euid)))) fatal ("Unable to look up user name");
                                        /* init environment if not root */
    if (euid) env_init (pw->pw_name,((s=getenv("HOME")) && *s) ? s:pw->pw_dir);
    else ret = pw->pw_name;             /* in case UID 0 user is other than root */
  }
  if (myUserName) {                     /* logged in? */
    if (flags) *flags = anonymous ? MU_ANONYMOUS : MU_LOGGEDIN;
    ret = myUserName;                   /* return user name */
  }
  else if (flags) *flags = MU_NOTLOGGEDIN;
/******Added for debugging **********/
printf("Myusername is %s \n", ret);

  return ret;
}


/* Return my local host name
 * Returns: my local host name
 */

char *mylocalhost ()
{
  char tmp[MAILTMPLEN];
  struct hostent *host_name;
  if (!myLocalHost) {
    gethostname(tmp,MAILTMPLEN);/* get local host name */
    myLocalHost = cpystr ((host_name = gethostbyname (tmp)) ?
                              host_name->h_name : tmp);
  }
  return myLocalHost;
}
```

```
/* Return my home directory name
 * Returns: my home directory name
 */

char *myhomedir ()
{
/*****************************
         Added Code
 *****************************
  access_ma *user_level;  /*Mandatory access type holds MA parameters
  char *null_path = NIL;  /*Used to supply NIL to function getlevel()
  char *add_dir = '\0';       /*Used to store additional path name
  char secret[] = "secret";
  char conf[]   = "conf";
  char unclass[]= "unclass";
  char tmp[MAILTMPLEN];
/*****************************/

  if (!myHomeDir) myusername ();
/*  {
/*******************************************************************
         Added Code
 *******************************************************************

  getlevel(null_path, user_level);
  if(user_level->security_level==2) /* Are we at the secret level/
  {
    add_dir = secret;  /*Add the secret path*
  }
  else if(user_level->security_level==1) /* Are we at the confidential level?*
  {
    add_dir = conf;   /*Add the conf path*
  }
  else  /* Everything else is unclassified/
  {
    add_dir = unclass; /*Add the unclass path*
  }
  printf("Adding additional myhomedir path name of %s\n", add_dir);
   strcat (tmp, add_dir);  /* Add additional dir to path*
   strcat (tmp,myusername ());/* initialize if first time *
   *myHomeDir = *tmp;
  }
/*************************************************************************/

 return myHomeDir ? myHomeDir : "";
}


/* Return system standard INBOX
 * Accepts: buffer string
 */

char *sysinbox ()
{
 char tmp[MAILTMPLEN];
 if (!sysInbox) {               /* initialize if first time */
  sprintf (tmp,"%s/%s",MAILSPOOL,myusername ());
  sysInbox = cpystr (tmp);           /* system inbox is from mail spool */
 }
 return sysInbox;
}


/* Return mailbox directory name
 * Accepts: destination buffer
 *          directory prefix
 *          name in directory
 * Returns: file name or NIL if error
 */

char *mailboxdir (char *dst,char *dir,char *name)
{
/******************************
        Added Code
 ******************************
  access_ma *user_level;  /*Mandatory access type holds MA parameters*
  char *null_path = NIL;  /*Used to supply NIL to function getlevel()*
  char *add_dir = '\0';       /*Used to store additional path name*
  char secret[] = "secret";
  char conf[]   = "conf";
  char unclass[]= "unclass";
/******************************/

 char tmp[MAILTMPLEN];
 if (dir || name) {              /* if either argument provided */
  if (dir) strcpy (tmp,dir); /* write directory prefix */
  else tmp[0] = '\0';                    /* otherwise null string */
/**********************************************************************
        Added Code
 **********************************************************************

  getlevel(null_path, user_level);
  if(user_level->security_level==2) /* Are we at the secret level?*
  {
    add_dir = secret;  /*Add the secret path*
  }
  else if(user_level->security_level==1) /* Are we at the confidential level?*
  {
    add_dir = conf;   /*Add the conf path*
  }
  else  /* Everything else is unclassified*
  {
    add_dir = unclass; /*Add the unclass path*
  }
  printf("Adding additional mailboxdir path name of %s\n", add_dir);
   strcat (tmp, add_dir);  /* Add additional dir to path


/*****************************************************************************/

   if (name) strcat (tmp,name);/* write name in directory */
                                    /* validate name, return its name */
   if (!mailboxfile (dst,tmp)) return NIL;
 }
 else strcpy (dst,myhomedir());/* no arguments, wants home directory */
 return dst;                        /* return the name */
}

/* Return mailbox file name
 * Accepts: destination buffer
 *          mailbox name
 * Returns: file name or empty string for driver-selected INBOX or NIL if error
 */
char *mailboxfile (char *dst,char *name)
{
 struct passwd *pw;
 char *dir = myhomedir ();
 *dst = '\0';                       /* default to empty string */
                                    /* check invalid name */
 if (!name || !*name || (*name == '{')) return NIL;
                                    /* check for INBOX */
 if (((name[0] == 'I') || (name[0] == 'i')) &&
    ((name[1] == 'N') || (name[1] == 'n')) &&
    ((name[2] == 'B') || (name[2] == 'b')) &&
    ((name[3] == 'O') || (name[3] == 'o')) &&
    ((name[4] == 'X') || (name[4] == 'x')) && !name[5]) {
                                    /* if restricted, canonicalize name of INBOX */
   if (anonymous || blackBox) name = "INBOX";
   else return dst;            /* else driver selects the INBOX name */
 }
                                    /* restricted name? */
 else if ((*name == '#') || anonymous || blackBox) {
   if (strstr (name,"..") || strstr (name,"//") || strstr (name,"/~"))
     return NIL;               /* none of these allowed when restricted */
   switch (*name) {                 /* what kind of restricted name? */
   case '#':                        /* namespace name */
     if (((name[1] == 'f') || (name[1] == 'F')) &&
          ((name[2] == 't') || (name[2] == 'T')) &&
          ((name[3] == 'p') || (name[3] == 'P')) &&
          (name[4] == '/') && (dir = ftpHome)) name += 5;
     else if (((name[1] == 'p') || (name[1] == 'P')) &&
             ((name[2] == 'u') || (name[2] == 'U')) &&
                ((name[3] == 'b') || (name[3] == 'B')) &&
                ((name[4] == 'l') || (name[4] == 'L')) &&
                ((name[5] == 'i') || (name[5] == 'I')) &&
                ((name[6] == 'c') || (name[6] == 'C')) &&
                (name[7] == '/') && (dir = publicHome)) name += 8;
       else if (!anonymous && ((name[1] == 's') || (name[1] == 'S')) &&
                ((name[2] == 'h') || (name[2] == 'H')) &&
                ((name[3] == 'a') || (name[3] == 'A')) &&
                ((name[4] == 'r') || (name[4] == 'R')) &&
                ((name[5] == 'e') || (name[5] == 'E')) &&
                ((name[6] == 'd') || (name[6] == 'D')) &&
                (name[7] == '/') && (dir = sharedHome)) name += 8;
       else return NIL;              /* unknown namespace name */
       break;
      case '/':                      /* rooted restricted name */
       if (anonymous) return NIL;/* anonymous can't do this */
       dir = blackBoxDir;     /* base is black box directory */
       name++;                       /* skip past delimiter */
       break;
      }
    }
                                     /* absolute path name? */
    else if (*name == '/') return strcpy (dst,name);
                                     /* some home directory? */
    else if ((*name == '~') && *++name) {
     if (*name == '/') name++;          /* yes, my home directory? */
     else {                    /* no, copy user name */
       for (dir = dst; *name && (*name != '/'); *dir++ = *name++);
       *dir++ = '\0';             /* tie off user name, look up in passwd file */
       if (!((pw = getpwnam (dst)) && (dir = pw->pw_dir))) return NIL;
       if (*name) *name++;  /* skip past the slash */
     }
    }
                                     /* build resulting name */
    sprintf (dst,"%s/%s",dir,name); /*Added tmp2 here*/
    return dst;                      /* return it */
   }
```

```
/* Lock file name
 * Accepts: scratch buffer
 *          file name
 * Returns: file descriptor of lock or -1 if error
 */

int lockname (char *lock,char *fname)
{
 char *s = strrchr (fname,'/');
 struct stat sbuf;
 if (stat (fname,&sbuf))    /* get file status */
  sprintf (lock,"/tmp/.%s",s ? s : fname);
 else locksbuf (lock,(void *) &sbuf);
 return lock_work (lock);
}


/* Lock file descriptor
 * Accepts: file descriptor
 *          lock file name buffer
 *          type of locking operation (LOCK_SH or LOCK_EX)
 * Returns: file descriptor of lock or -1 if failure
 */

int lockfd (int fd,char *lock,int op)
{
 int ld;
 struct stat sbuf;
                                    /* get data for this file */
 if (fstat (fd,&sbuf)) return -1;
 locksbuf (lock,(void *) &sbuf);
                                    /* get the lock */
 while (((ld = lock_work (lock)) < 0) && (errno == EEXIST));
 if (ld >= 0) flock (ld,op); /* did we get it? */
 else syslog (LOG_INFO,"Mailbox lock file %s open failure: %s",lock,
              strerror (errno));
 return ld;                         /* return locking file descriptor */
}


/* Make temporary lock file name
 * Accepts: lock file name buffer
 *          pointer to stat() buffer
 */

void locksbuf (char *lock,void *sb)
{
 struct stat *sbuf = (struct stat *) sb;
                                    /* make temporary lock file name */
 sprintf (lock,"/tmp/.%lx.%lx",(long) sbuf->st_dev,(long) sbuf->st_ino);
}

/* Lock file name worker
 * Accepts: lock file name
 * Returns: file descriptor or -1 if error
 */

int lock_work (char *lock)
{
  int fd;
  long nlinks = chk_notsymlink (lock);
  if (!nlinks) return -1;       /* fail if symbolic link */
  if (nlinks > 1) {             /* extra hard link to the file? */
    mm_log ("SECURITY ALERT: hard link to lock name!",ERROR);
    syslog (LOG_CRIT,"SECURITY PROBLEM: lock file %s has a hard link",lock);
    return -1;
  }
  if ((fd = open (lock,O_RDWR | O_CREAT | ((nlinks < 0) ? O_EXCL : NIL),
                  (int) mail_parameters (NIL,GET_LOCKPROTECTION,NIL))) >= 0)
                                /* make sure mode OK (don't use fchmod()) */
    chmod (lock,(int) mail_parameters (NIL,GET_LOCKPROTECTION,NIL));
  return fd;
}


/* Check to make sure not a symlink
 * Accepts: file name
 * Returns: -1 if doesn't exist, NIL if symlink, else number of hard links
 */

long chk_notsymlink (char *name)
{
  struct stat sbuf;
                                /* name exists? */
  if (lstat (name,&sbuf)) return -1;
                                /* forbid symbolic link */
  if ((sbuf.st_mode & S_IFMT) == S_IFLNK) {
    mm_log ("SECURITY ALERT: symbolic link on lock name!",ERROR);
    syslog (LOG_CRIT,"SECURITY PROBLEM: lock file %s is a symbolic link",name);
    return NIL;
  }
  return (long) sbuf.st_nlink;  /* return number of hard links */
}


/* Unlock file descriptor
 * Accepts: file descriptor
 *          lock file name from lockfd()
 */

void unlockfd (int fd,char *lock)
```

```
{
                                /* delete the file if no sharers */
 if (!flock (fd,LOCK_EX|LOCK_NB)) unlink (lock);
 flock (fd,LOCK_UN);            /* unlock it */
 close (fd);                    /* close it */
}

/* Determine default prototype stream to user
 * Accepts: type (NIL for create, T for append)
 * Returns: default prototype stream
 */

MAILSTREAM *default_proto (long type)
{
 myusername ();          /* make sure initialized */
                                /* return default driver's prototype */
 return type ? appendProto : createProto;
}


/* Set up user flags for stream
 * Accepts: MAIL stream
 * Returns: MAIL stream with user flags set up
 */

MAILSTREAM *user_flags (MAILSTREAM *stream)
{
 int i;
 myusername ();          /* make sure initialized */
 for (i = 0; i < NUSERFLAGS && userFlags[i]; ++i)
   if (!stream->user_flags[i]) stream->user_flags[i] = cpystr (userFlags[i]);
 return stream;
}


/* Return nth user flag
 * Accepts: user flag number
 * Returns: flag
 */

char *default_user_flag (unsigned long i)
{
 myusername ();          /* make sure initialized */
 return userFlags[i];
}
```

```
/* Process rc file
 * Accepts: file name
 *          .mminit flag
 * Don't even think of using this feature.
 */

void dorc (char *file,long flag)
{
 int i;
 char *s,*t,*k,tmp[MAILTMPLEN],tmpx[MAILTMPLEN];
 extern MAILSTREAM STDPROTO;
 DRIVER *d;
 FILE *f = fopen (file,"r");
                                   /* no file or ill-advised usage */
 if (!(f && (s = fgets (tmp,MAILTMPLEN,f)) && (t = strchr (s,'\n')) &&
        (flag ||
         (!strcmp (s,"I accept the risk for IMAP toolkit 4.1.\n") &&
          (s = fgets (tmp,MAILTMPLEN,f)) && (t = strchr (s,'\n')))))) return;
 do {
  *t++ = '\0';                /* tie off line, find second space */
  if ((k = strchr (s,' ')) && (k = strchr (++k,' '))) {
   *k++ = '\0';               /* tie off two words*/
   lcase (s);                 /* make case-independent */
   if (!(userFlags[0] || strcmp (s,"set keywords"))) {
        k = strtok (k,", ");/* yes, get first keyword */
                                   /* copy keyword list */
        for (i = 0; k && i < NUSERFLAGS; ++i) {
         if (userFlags[i]) fs_give ((void **) &userFlags[i]);
         userFlags[i] = cpystr (k);
         k = strtok (NIL,", ");
        }
   }
   else if (!flag) {               /* none of these valid in .mminit */
```

```
if (!(createProto || strcmp (s,"set new-folder-format"))) {
  if (!strcmp (lcase (k),"same-as-inbox"))
    createProto = ((d = mail_valid (NIL,"INBOX",NIL)) &&
                        strcmp (d->name,"dummy")) ?
                         ((*d->open) (NIL)) : &STDPROTO;
  else if (!strcmp (k,"system-standard")) createProto = &STDPROTO;
  else {              /* see if a driver name */
    for (d = (DRIVER *) mail_parameters (NIL,GET_DRIVERS,NIL);
         d && strcmp (d->name,k); d = d->next);
    if (d) createProto = (*d->open) (NIL);
    else {            /* duh... */
      sprintf (tmpx,"Unknown new folder format in %s: %s",file,k);
      mm_log (tmpx,WARN);
    }
  }
}
if (!(appendProto || strcmp (s,"set empty-folder-format"))) {
  if (!strcmp (lcase (k),"same-as-inbox"))
    appendProto = ((d = mail_valid (NIL,"INBOX",NIL)) &&
                        strcmp (d->name,"dummy")) ?
                         ((*d->open) (NIL)) : &STDPROTO;
  else if (!strcmp (k,"system-standard")) appendProto = &STDPROTO;
  else {              /* see if a driver name */
    for (d = (DRIVER *) mail_parameters (NIL,GET_DRIVERS,NIL);
         d && strcmp (d->name,k); d = d->next);
    if (d) appendProto = (*d->open) (NIL);
    else {            /* duh... */
      sprintf (tmpx,"Unknown empty folder format in %s: %s",file,k);
      mm_log (tmpx,WARN);
    }
  }
}
else if (!strcmp (s,"set from-widget"))
  mail_parameters (NIL,SET_FROMWIDGET,strcmp (lcase (k),"header-only")
                        ? (void *) T : NIL);
else if (!strcmp (s,"set black-box-directory")) {
  if (blackBoxDir)            /* users aren't allowed to do this */
    mm_log ("Can't set black-box-directory in user init",ERROR);
  else blackBoxDir = cpystr (k);
}
else if (!strcmp (s,"set black-box-default-home-directory")) {
  if (!blackBoxDir)
    mm_log ("Must set black-box-directory before default-home",ERROR);
  else if (blackBoxDefaultHome)
    mm_log ("Can't set black-box-default-home-directory in user init",
            ERROR);
  else blackBoxDefaultHome = cpystr (k);
}
```

```
else if (!strcmp (s,"set local-host")) {
  fs_give ((void **) &myLocalHost);
  myLocalHost = cpystr (k);
}
else if (!strcmp (s,"set news-active-file")) {
  fs_give ((void **) &newsActive);
  newsActive = cpystr (k);
}
else if (!strcmp (s,"set news-spool-directory")) {
  fs_give ((void **) &newsSpool);
  newsSpool = cpystr (k);
}
else if (!strcmp (s,"set news-state-file")) {
  fs_give ((void **) &myNewsrc);
  myNewsrc = cpystr (k);
}
else if (!strcmp (s,"set anonymous-home-directory")) {
  fs_give ((void **) &anonymousHome);
  anonymousHome = cpystr (k);
}
else if (!strcmp (s,"set ftp-export-directory")) {
  fs_give ((void **) &ftpHome);
  ftpHome = cpystr (k);
}
else if (!strcmp (s,"set public-home-directory")) {
  fs_give ((void **) &publicHome);
  publicHome = cpystr (k);
}
else if (!strcmp (s,"set shared-home-directory")) {
  fs_give ((void **) &sharedHome);
  sharedHome = cpystr (k);
}
else if (!strcmp (s,"set system-inbox")) {
  fs_give ((void **) &sysInbox);
  sysInbox = cpystr (k);
}
else if (!strcmp (s,"set rsh-command"))
  mail_parameters (NIL,SET_RSHCOMMAND,(void *) k);
else if (!strcmp (s,"set rsh-path"))
  mail_parameters (NIL,SET_RSHPATH,(void *) k);
```

```
        else if (!strcmp (s,"set tcp-open-timeout"))
          mail_parameters (NIL,SET_OPENTIMEOUT,(void *) atol (k));
        else if (!strcmp (s,"set tcp-read-timeout"))
          mail_parameters (NIL,SET_READTIMEOUT,(void *) atol (k));
        else if (!strcmp (s,"set tcp-write-timeout"))
          mail_parameters (NIL,SET_WRITETIMEOUT,(void *) atol (k));
        else if (!strcmp (s,"set rsh-timeout"))
          mail_parameters (NIL,SET_RSHTIMEOUT,(void *) atol (k));
        else if (!strcmp (s,"set maximum-login-trials"))
          mail_parameters (NIL,SET_MAXLOGINTRIALS,(void *) atol (k));
        else if (!strcmp (s,"set lookahead"))
          mail_parameters (NIL,SET_LOOKAHEAD,(void *) atol (k));
        else if (!strcmp (s,"set prefetch"))
          mail_parameters (NIL,SET_PREFETCH,(void *) atol (k));
        else if (!strcmp (s,"set close-on-error"))
          mail_parameters (NIL,SET_CLOSEONERROR,(void *) atol (k));
        else if (!strcmp (s,"set imap-port"))
          mail_parameters (NIL,SET_IMAPPORT,(void *) atol (k));
        else if (!strcmp (s,"set pop3-port"))
          mail_parameters (NIL,SET_POP3PORT,(void *) atol (k));
        else if (!strcmp (s,"set uid-lookahead"))
          mail_parameters (NIL,SET_UIDLOOKAHEAD,(void *) atol (k));
        else if (!strcmp (s,"set mailbox-protection"))
          mail_parameters (NIL,SET_MBXPROTECTION,(void *) atol (k));
        else if (!strcmp (s,"set directory-protection"))
          mail_parameters (NIL,SET_DIRPROTECTION,(void *) atol (k));
        else if (!strcmp (s,"set lock-protection"))
          mail_parameters (NIL,SET_LOCKPROTECTION,(void *) atol (k));
        else if (!strcmp (s,"set disable-fcntl-locking"))
          mail_parameters (NIL,SET_DISABLEFCNTLLOCK,(void *) atol (k));
        else if (!strcmp (s,"set lock-eacces-error"))
          mail_parameters (NIL,SET_LOCKEACCESERROR,(void *) atol (k));
        else if (!strcmp (s,"set list-maximum-level"))
          mail_parameters (NIL,SET_LISTMAXLEVEL,(void *) atol (k));
        else if (!strcmp (s,"set allowed-login-attempts")) logtry = atoi (k);
    }
   }
  }
  while ((s = fgets (tmp,MAILTMPLEN,f)) && (t = strchr (s,'\n')));
  fclose (f);                     /* flush the file */
}
```

```
/* INBOX create function for tmail/dmail use only
 * Accepts: mail stream
 *            path name buffer, preloaded with driver-dependent path
 * Returns: T on success, NIL on failure
 *
 * This routine is evil and a truly incredible kludge.  It is private for
 * tmail/dmail and is not supported for any other application.
 */

long path_create (MAILSTREAM *stream,char *path)
{
  long ret;
                                /* do the easy thing if not a black box */
  if (!blackBox) return mail_create (stream,path);
                                /* toss out driver dependent names */
  printf (path,"%s/INBOX",myhomedir ());
  blackBox = NIL;               /* well that's evil - evil is going on */
  ret = mail_create (stream,path);
  blackBox = T;                 /* restore the box */
  return ret;
}
```

OS_XTS.H

```
/* Filename:      os_xts.h
 *
 * Program:       Operating-system dependent routines -- XTS-300 version
 *
 * Author:        Mark Crispin
 *                Networks and Distributed Computing
 *                Computing & Communications
 *                University of Washington
 *                Administration Building, AG-44
 *                Seattle, WA  98195
 *                Internet: MRC@CAC.Washington.EDU
 *
 * Date: 10 April 1992
 *
 * Modified by: Brad Eads
 *                Naval Postgraduate School
 *                Monterey CA 93940
 *                17 Jan 1999
 *
 * Last Edited:   27 Feb 1999
 *
 * Copyright 1998 by the University of Washington
 *
 *  Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose and without fee is hereby granted, provided
 * that the above copyright notice appears in all copies and that both the
 * above copyright notice and this permission notice appear in supporting
 * documentation, and that the name of the University of Washington not be
 * used in advertising or publicity pertaining to distribution of the software
 * without specific, written prior permission.  This software is made
 * available "as is", and
 * THE UNIVERSITY OF WASHINGTON DISCLAIMS ALL WARRANTIES, EXPRESS OR
IMPLIED,
 * WITH REGARD TO THIS SOFTWARE, INCLUDING WITHOUT LIMITATION ALL IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND
IN
 * NO EVENT SHALL THE UNIVERSITY OF WASHINGTON BE LIABLE FOR ANY SPECIAL,
 * INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER
RESULTING FROM
 * LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, TORT
 * (INCLUDING NEGLIGENCE) OR STRICT LIABILITY, ARISING OUT OF OR IN CONNECTION
 * WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */
#include <string.h>
#include <sys/types.h>
#include <stdlib.h>         /***Added for XTS-300***/
#include <sys/dirent.h>     /***Added for XTS-300***/
#include <fcntl.h>
#include <unistd.h>
```

```
#include <time.h>           /***Added for XTS-300***/
#include <utime.h>          /***Added for XTS-300***/
#define char void
#include <memory.h>
#undef char
#include <sys/dir.h>
#include <sys/file.h>
#include <ustat.h>
/****************************************
        Added for the multilevel system
****************************************/
#include <level.h>

/***************************************/

/* Many versions of SysV get this wrong */

#define setpgrp(a,b) Setpgrp(a,b)


/* Different names between BSD and System V */
/* ifndef added to facilitate compiling on the XTS*/
#ifndef L_SET
#define L_SET SEEK_SET
#endif
#ifndef L_INCR
#define L_INCR SEEK_CUR
#endif
#ifndef L_XTND
#define L_XTND SEEK_END
#endif
#ifndef lstat
#define lstat stat
#endif
#ifndef random
#define random lrand48
#endif
#ifndef SIGSTOP
#define SIGSTOP SIGQUIT
#endif

#define S_IFLNK 0120000


/* syslog() emulation */
/* None of these work on the xts, logging stubbed out*/
/****************************************************/

#define LOG_MAIL       (2<<3)  /* mail system */
#define LOG_AUTH       (4<<3)  /* security/authorization messages */
#define LOG_ALERT      (5<<3)
```

```
#define LOG_INFO        (6<<3)
#define LOG_CRIT        (7<<3)
#define LOG_ERR                 (8<<3)
#define LOG_PID                 (9<<3)
/*** End stubbed loging commands ***/

/* For flock() emulation */

#define flock bsd_flock

#define LOCK_SH 1
#define LOCK_EX 2
#define LOCK_NB 4
#define LOCK_UN 8


/* For setitimer() emulation */

#define ITIMER_REAL    0


/* For opendir() emulation */

typedef struct _dirdesc {
  int dd_fd;
  long dd_loc;
  long dd_size;
  char *dd_buf;
} DIR;

#include "env_unix.h"
#include "fs.h"
#include "ftl.h"
#include "nl.h"
#include "tcp.h"
#include "lockfix.h"


long gethostid (void);
DIR *opendir (char * name);
int closedir (DIR *d);
struct direct *readdir (DIR *d);
typedef int (*select_t) (struct direct *name);
typedef int (*compar_t) (void *d1,void *d2);
int scandir (char *dirname,struct direct ***namelist,select_t select,
                compar_t compar);
int bsd_flock (int fd,int operation);
```

OS_XTS.C

```
/* Filename:    os_xts.c
 *
 * Program:     Operating-system dependent routines -- XTS-300 Version
 *
 * Author:      Mark Crispin
 *              Networks and Distributed Computing
 *              Computing & Communications
 *              University of Washington
 *              Administration Building, AG-44
 *              Seattle, WA  98195
 *              Internet: MRC@CAC.Washington.EDU
 *
 * Date:  10 April 1992
 *
 * Modified by: Brad Eads
 *              Naval Postgraduate School
 *              Monterey CA 93940
 *              17 Jan 1999
 *
 * Last Edited:  27 Feb 1999
 *
 * Copyright 1997 by the University of Washington
 *
 *  Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose and without fee is hereby granted, provided
 * that the above copyright notice appears in all copies and that both the
 * above copyright notice and this permission notice appear in supporting
 * documentation, and that the name of the University of Washington not be
 * used in advertising or publicity pertaining to distribution of the software
 * without specific, written prior permission.  This software is made
 * available "as is", and
 * THE UNIVERSITY OF WASHINGTON DISCLAIMS ALL WARRANTIES, EXPRESS OR
IMPLIED,
 * WITH REGARD TO THIS SOFTWARE, INCLUDING WITHOUT LIMITATION ALL IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND
IN
 * NO EVENT SHALL THE UNIVERSITY OF WASHINGTON BE LIABLE FOR ANY SPECIAL,
 * INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER
RESULTING FROM
 * LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, TORT
 * (INCLUDING NEGLIGENCE) OR STRICT LIABILITY, ARISING OUT OF OR IN CONNECTION
 * WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

#include "tcp_unix.h"           /* must be before osdep includes tcp.h */
#include "mail.h"
#include "osdep.h"
#include <ctype.h>
```

```
#include <stdio.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <errno.h>
extern int errno;
#include <pwd.h>
#include <grp.h>
#include <sys/socket.h>
#include <time.h>
#define KERNEL
#include <sys/time.h>
#undef KERNEL
#include "misc.h"

#define DIR_SIZE(d) sizeof (DIR)

extern int sys_nerr;
extern char *sys_errlist[];

#define toint(c)    ((c)-'0')
#define isodigit(c)         (((unsigned)(c)>=060)&((unsigned)(c)<=067))

#define  NBBY   8           /* number of bits in a byte */
#define  FD_SETSIZE    256

typedef long        fd_mask;
#define NFDBITS             (sizeof(fd_mask) * NBBY)
                                         /* bits per mask */
#define  howmany(x, y)   (((x)+((y)-1))/(y))

typedef  struct fd_set {
  fd_mask fds_bits[howmany(FD_SETSIZE, NFDBITS)];
} fd_set;

#define  FD_SET(n, p)     ((p)->fds_bits[(n)/NFDBITS] |= \
                                         (1 << ((n) % NFDBITS)))
#define  FD_CLR(n, p)     ((p)->fds_bits[(n)/NFDBITS] &= \
                                         ~(1 << ((n) % NFDBITS)))
#define  FD_ISSET(n, p)  ((p)->fds_bits[(n)/NFDBITS] & \
                                         (1 << ((n) % NFDBITS)))
#define FD_ZERO(p)       bzero((char *)(p), sizeof(*(p)))

#include "fs_unix.c"
#include "ftl_unix.c"
#include "nl_unix.c"
#include "env_unix.c"
#include "tcp_unix.c"
#include "gr_wait.c"
#undef flock
```

```
#include "flockxts.c"
#include "opendir.c"
#include "scandir.c"
#include "memmove2.c"
#include "strstr.c"
#include "strerror.c"
#include "strtoul.c"
#include "tz_xts.c"
#include "gethstid.c"
#include "fsync.c"
#undef setpgrp
#include "setpgrp.c"

/* Emulator for BSD syslog() routine stubbed out for XTS
 * Accepts: priority
 *          message
 *          parameters
 */

int syslog (int priority,char *message,char *parameters)
{
        return(1);
}

/* Emulator for BSD openlog() routine stubbed out for XTS
 * Accepts: identity
 *          options
 *          facility
 */

int openlog (char *ident,int logopt,int facility)
{
        return(1);
}

/* Emulator for BSD ftruncate() routine
 * Accepts: file descriptor
 *          length
 */

int ftruncate (int fd,long length)
{
        struct flock fb;
        fb.l_whence = 0;
        fb.l_len    = 0;
        fb.l_start  = length;
        fb.l_type   = F_WRLCK;  /* ativate the writelock on the file space*/
        return (1);
} /* end ftruncate*/
```

```
int initgroups (const char* grp_ptr, gid_t grp_id)
{
        return(1);
}
```

TZ_XTS.C

```
/* Filename: tz_xts.c
 *
 * Program:  XTS-300 time zone string
 *
 * Author:         Brad Eads
 *                 Naval Postgraduate School
 *                 Based on the original work of Mark Crispin
 *                 University of Washington
 *
 * Dated:          22 Jan 1999
 * Last edited: 27 Feb 1999
 *
 */


/* Appends the local timezone name to the input destination string
*/

void rfc822_timezone (char *s, void *t)
{
        tzset();  /* Gets the timezone information */
        sprintf (s + strlen(s), " (%s)", ((*((struct tm*)t)).tm_isdst > 0) ? "DST" : " ");
}
```

FLOCKXTS.C

```
/* Filename:     flockxts.c
 *
 * Program:      File lock modified for use on XTS-300
 *
 * Author:       Mark Crispin
 *               Networks and Distributed Computing
 *               Computing & Communications
 *               University of Washington
 *               Administration Building, AG-44
 *               Seattle, WA  98195
 *               Internet: MRC@CAC.Washington.EDU
 *
 * Date: 1 August 1988
 * Modified by: Brad Eads
 *               Naval Postgraduate School
 *               Center for Information Systems Security Studies and Research
 *               Monterey CA 93940
 *
 * Reason for modification:  This file would not compile on the XTS-300 "&fl"
 *                           in the return line had to be cast to "(in)&fl".
 *
 * Last Edited:  30 Jan 1999
 *
 * Copyright 1998 by the University of Washington
 *
 *  Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose and without fee is hereby granted, provided
 * that the above copyright notice appears in all copies and that both the
 * above copyright notice and this permission notice appear in supporting
 * documentation, and that the name of the University of Washington not be
 * used in advertising or publicity pertaining to distribution of the software
 * without specific, written prior permission.  This software is made available
 * "as is", and
 * THE UNIVERSITY OF WASHINGTON DISCLAIMS ALL WARRANTIES, EXPRESS OR
IMPLIED,
 * WITH REGARD TO THIS SOFTWARE, INCLUDING WITHOUT LIMITATION ALL IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND
IN
 * NO EVENT SHALL THE UNIVERSITY OF WASHINGTON BE LIABLE FOR ANY SPECIAL,
 * INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER
RESULTING FROM
 * LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, TORT
 * (INCLUDING NEGLIGENCE) OR STRICT LIABILITY, ARISING OUT OF OR IN CONNECTION
 * WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 */

#define flock fcntl_flock
#include "flcksafe.c"            /* get safe locking routines */
```

```
#undef flock
#include "flockbsd.c"                /* get flock() NFS jacket */

/* Emulator for flock() call using fcntl() locking
 * Accepts: file descriptor
 *          operation bitmask
 * Returns: 0 if successful, -1 if failure
 */

int fcntl_flock (int fd,int op)
{
 struct flock fl;
                                     /* lock applies to entire file */
 fl.l_whence = fl.l_start = fl.l_len = 0;
 fl.l_pid = getpid ();               /* shouldn't be necessary */
 switch (op & ~LOCK_NB) {            /* translate to fcntl() operation */
 case LOCK_EX:                               /* exclusive */
  fl.l_type = F_WRLCK;
  break;
 case LOCK_SH:                               /* shared */
  fl.l_type = F_RDLCK;
  break;
 case LOCK_UN:                               /* unlock */
  fl.l_type = F_UNLCK;
  break;
 default:                   /* default */
  errno = EINVAL;
  return -1;
 }
/*****************************************************************************
** Changed &fl to (int)&fl in the next line
*****************************************************************************/
 return (fcntl (fd,(op & LOCK_NB) ? F_SETLK : F_SETLKW, (int)&fl) == -1) ? -1 : 0;
}
```

LIST OF REFERENCES

[1] *Multilevel Security in the Department of Defense: The Basics*, Defense Information Systems Agency, Department of Defense Multilevel Security Program, 1 March 1995.

[2] *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1985.

[3] Multilevel Security Program, Defense Information Systems Agency, http://www.disa.mil/line/mlshome.html, 2 April 1996.

[4] Kang, M. H., A. P. Moore, and I. S. Moskowitz. "Design and Assurance Strategy for the NRL Pump." Computer 31,4 (April 1998), pp. 56-64.

[5] Information Diode Executive Summary, Galaxy Computer Services Incorporated, Santa Fe, NM 87501

[6] Bell, David E. and Leonard LaPadula. Secure Computer Systems: Mathematical Foundations and Model. Technical Report M74-244, MITRE Corp., Bedford, MA, 1973.

[7] Anderson, James P., Computer Security Technology Planning Study, ESD-TR-73-51, Vol. I, October 1972.

[8] *Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments*, CSC-STD-004-85, 25 June 1985.

[9] Downey, James P. and Dion A. Robb, *Design of a High Assurance, Multilevel Secure Mail Server,* Masters Thesis, Naval Postgraduate School, September 1997.

[10] BryerJoyner, Susan, and Scott Heller, *Initiating a Secure Session Between a Hign Assurance Server and a Trusted Computing Base Extension Over an Untrusted Local Area Network*, Masters Thesis, Naval Postgraduate School, March 1999.

[11] Hackerson, Jason X., *Design of a Trusted Computing Base Extension for Commercial Off-The-Shelf Workstations (TCBE),* Masters Thesis, Naval Postgraduate School, September 1998.

[12] *Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments*, CSC-STD-004-85, 25 June 1985.

[13] Wang Government Services, Inc., 7900 Westpark Drive, McLean, VA 22102, *XTS-300 User's Manual,* STOP 4.4.2 Version, March 1998.

[14] Crocker, David H., *Standard for the Format of ARPA Internet Text Messages*, RFC 822, August 13, 1982.

[15] Postel, Jonathan B., *Simple Mail Transfer Protocol*, RFC 821, August 1982.

[16] Rhoton, John, *X.400 and SMTP Battle of the E-mail Protocols*, Digital Press, 1997, pp. 88-100.

[17] Myers, J., RFC 1939, *Post Office Protocol – Version 3*, May 1996.

[18] Crispin, M., RFC 2060, *Internet Message Access Protocol –Version4rev1*, December 1996.

[19] www.washington.edu/imap/

[20] Wang Government Services, Inc., 7900 Westpark Drive, McLean, VA 22102, *XTS-300 Application Programmer's Manual*, STOP 4.4.2 Version, March 1998.

[21] Kang, Myong H., Judith N. Froscher, and Brian J. Eppinger, Towards an Infrastructure for MLS Distributed Computing, *Proceedings of the 14th Annual Computer Security Applications Conference*, December 1988, pp. 91-100.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.. 2
   8725 John J. Kingman Road, Ste 0944
   Fort Belvoir, VA 22060-6218

2. Dudley Knox Library.. 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Director, Training and Education.. 1
   MCCDC, Code C46
   1019 Elliot Road
   Quantico, VA 22134-5027

4. Director, Marine Corps Research Center.. 2
   MCCDC, Code C40RC
   2040 Broadway Street
   Quantico, VA 22134-5107

5. Director, Studies and Analysis Division.. 1
   MCCDC, Code C45
   300 Russell Road
   Quantico, VA 22134-5130

6. Marine Corps Representative.. 1
   Naval Postgraduate School
   Code 037, Bldg. 234, HA-220
   699 Dyer Road
   Monterey, CA 93943

7. Chairman, Code CS.. 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

8. Dr. Cynthia E. Irvine.. 1
   Computer Science Department Code CS/Ic
   Naval Postgraduate School
   Monterey, CA 93943

9. Daniel F.Warren..1
Computer Science Department Code CS/W
Naval Postgraduate School
Monterey, CA 93943

10. Joseph O'Kane 1
National Security Agency
Research and Development Building
R23
9800 Savage Road
Fort Meade, MD 20755-6000

12. CAPT Dan Galik...1
Space and Naval Warfare Systems Command
PMW 161
Building OT-1, Room 1024
4301 Pacific Highway
San Diego, CA 92110-3127

13. Commander, Naval Security Group Command...1
Naval Security Group Headquarters
9800 Savage Road
Suite 6585
Fort Meade, MD 20755-6585
ATTN: Mr. James Shearer

14. Mr. George Bieber 1
Defense Information Systems Agency
Center for Information Systems Security
5113 Leesburg Pike, Suite 400
Falls Church, VA 22041-3230

15. CDR Chris Perry...1
N643
Presidential Tower 1
2511 South Jefferson Davis Highway
Arlington, VA 22202

16. Mr. James P. Anderson 1
James P. Anderson Co.
Box 42
Fort Washington, PA 19034

17. Mr. Charles Sherupski 1
Community CIO Office
Washington DC 20505

18. Ms. Deborah M. Cooper 1
Deborah M. Cooper Co.
P O Box 17753
Arlington VA 22216

19. DISA D6/IAESO/MSL Engineering.. 1
LCDR Downey, RM 639
5600 Columbia Pike
Falls Church, VA 22041-2717

20. Major Bradley R. Eads.. 1
582 Michelson Road
Unit D
Monterey, CA 93940